Analiza rada VGG arhitekture neuronske mreže

Trbara, Klara

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Klara Trbara

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

Dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Klara Trbara

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru Dr. sc. Tomislav Stipančić te obitelji i prijateljima na neizmjernoj potpori i strpljenju tijekom studiranja.

Klara Trbara

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: 602 – 04 / 22 – 6 / 1	
Ur.broj: 15 - 1703 - 22 -	

ZAVRŠNI ZADATAK

Student: **Klara Trbara** JMBAG: **0035211817**

Naslov rada na hrvatskom jeziku: **Analiza rada VGG arhitekture neuronske mreže**

Naslov rada na engleskom jeziku: **VGG neural network architecture performance analysis**

Opis zadatka:

U radu je potrebno razviti konvolucijsku neuronsku mrežu koja prepoznaje pse i mačke na slikama. Mrežu je potrebno trenirati i testirati koristeći bazu slika pasa i mačaka koja je slobodno dostupna za korištenje na Internetu.
U radu je potrebno:

- upoznati se sa strukturom baze slika pasa i mačaka,
- koristeći se pravilima dobre prakse pripremiti slike iz baze da budu prilagođene za neuronsku mrežu (prilagoditi veličine slika i njihovu orijentaciju),
- razviti osnovni model neuronske mreže VGG arhitekture blok po blok gdje će se pokazati kako raste preciznost klasifikacije mreže dodavanjem novih blokova,
- uvesti dodatna poboljšanja vezano za promjenu početnih parametara te dati kritički osvrt kako ona utječu na rezultate rada mreže,
- na istom skupu slika primijeniti poznatu arhitekturu neuronske mreže VGG-16 koja je unaprijed trenirana te usporediti rezultate s VGG arhitekturom koja sadrži tri bloka.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Kroz ovaj rad je prikazan razvoj konvolucijske neuronske mreže primjenjujući VGG arhitekturu u problemu rješavanja klasifikacije slika. Slike pasa i mačaka su zadane iz poznate baze podataka zadane u svrhu Kaggle-ovog natjecanja iz strojnog učenja iz 2013. godine. U uvodu rada se upoznajemo s osnovnom gradivnom jedinicom neuronske mreže perceptorom i osnovnom principima njezinog rada. U nastavku će se predstaviti opća arhitektura konvolucijskih neuronskih mreža. Naglasak ovog rada je na savladavanju principa razvoja osnovnog modela VGG arhitekture neuronske mreže s kojom se želi prikazati rast preciznosti klasifikacije mreže dodavanjem novih blokova. Prikazat će se razvoj osnovnog modela kroz korištenje *cloud* servisa i online sučelja *Google Colab* koristeći se otvorenim softverskim bibliotekama za strojno učenje (*Tenserflow*). Potom se na istoj bazi podataka testira već unaprijed istrenirana arhitektura neuronske mreže VGG-16 te se uspoređuju rezultati s osnovnim modelom.

Ključne riječi: konvolucija, CNN, VGG, umjetne neuronske mreže, VGG-16, perceptor, *Tensorflow*

SUMMARY

This paper presents the development of a convolutional neural network applying the VGG architecture to the problem of image classification. The images of dogs and cats were taken from a well-known database provided for Kaggle's 2013 machine learning competition.

In first part of the paper, we are introduced to the basic building block of the neural network, the preceptor, and the basic principles of its operation. Furthermore, the general architecture of convolutional neural networks will be presented.

The emphasis of this paper is on mastering the principles of developing the basic model of the VGG architecture of the neural network, with which we want to show the growth of the accuracy of the network classification by stacking new blocks. The development of the basic model will be presented through the use of cloud services and the Google Colab online interface using open software libraries for machine learning (TensorFlow). Then, the pre-trained VGG-16 neural network architecture is tested on the same database and the results are compared with the basic model.

Key words: convolution, CNN, VGG, artificial neural networks, VGG-16, preceptor, Tensorflow

1. UVOD

U današnjem vremenu gdje se jako veliki dio života odvija upravo na internetu te količina podatka koja danas cirkulira istim je u eksponencijalnom rastu jer se broj korisnika povećava iz dana u dan.

Sve te informacije su spremljene u razne baze podatka, u strukturiranom ili nestrukturiranom obliku, koje same za sebi ne znače puno no obrađene na pravi način mogu nam predvidjeti buduća ponašanja. Velike tvrtke su počele prepoznavati potencijal u tim generiranim podacima te ulažu ogromne količine novca u njihovu analizu kako bi prepoznali trend ponašanja svojih korisnika. Pri analizi takvih baza podatak uvelike im pomažu algoritmi strojnog učenja kojima je cilj kroz analizu prepoznati buduće ponašanje korisnika. Veliki iskorak prema naprijed u ovom području je uzrokovao razvoj dubokih neuronskih mreža.

Konvolucijske neuronske mreže su jedne od njih i koriste se pri klasifikaciji višedimenzionalnih baza podataka kao što su slike i video zapisi. Cilj ovog rada je bio upoznati se s osnovnim dijelovima algoritma konvolucijskih neuronskih mreža te ih primijeniti u razvoj osnovnog modela VGG strukture neuronske mreže koja će vršiti binarnu klasifikaciju nad poznatim bazom slika.

U prvom dijelu rada su obrađene osnovne gradivne jedinice neuronske mreže te osnovni algoritmi rada s kojima vrši transformacija ulaznih podataka. Nakon tog se upoznajemo s pojedinim slojevima konvolucijske neuronske mreže i detaljima vezanim za poboljšanje rada istih.

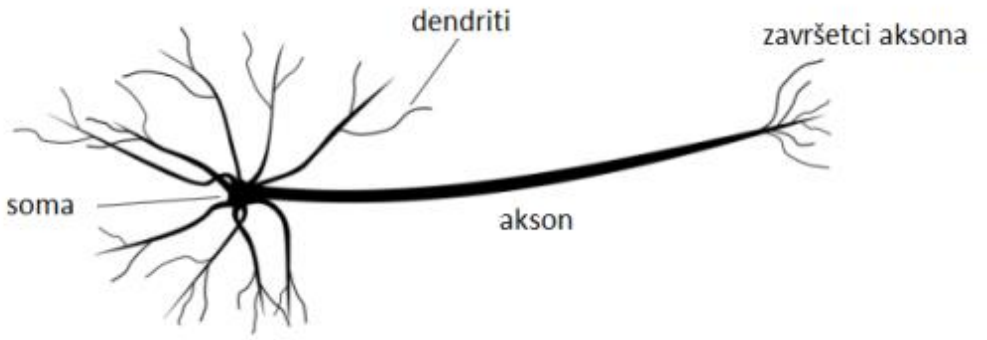
U drugom dijelu rada predstavlja se VGG arhitektura konvolucijske neuronske mreže u kojoj se metodom slaganja blokova razvija osnovni model. Kroz uvođenje metoda dodatnih poboljšanja uspoređuje se rad osnovnog modela s poboljšanim neuronskim mrežama. Nadalje se upoznajmo se pojmom prenesnog učenja kroz poznatu istreniranu strukturu neuronske mreže VGG-16.

2. Umjetne neuronske mreže

Ideja o stvaranju umjetnih neuronskih mreža započela je još u 40-im godinama prošlog stoljeća, a zasniva se na hipotezi utemeljenoj na mehanizmu neuroplastičnosti prozvanom Hebbijevo učenje. Teorija naglašava činjenicu ,ako stanice A djeluje na rad stanice B, ona mora izvršiti slanje signala malo prije stanice B, a ne u isto vrijeme. U svijetu umjetnih neuronskih mreža se to može shvatiti kao način prilagođavanja težina kod povezanih neurona.[1] Arhitektura umjetne neuronske mreže se sastoji od gradivnih jedinica prozvanim umjetni neuroni, koje su raspoređene u slojeve kroz koje prolaze naši ulazni podaci te se nad njima vrši određena transformacija. Cilj takve strukture je replicirati rad biološkog učenja.

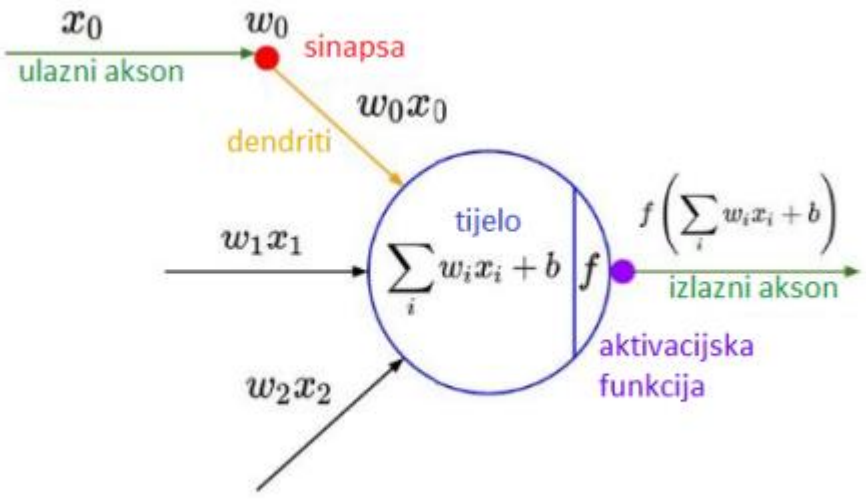
2.1. Modeliranje umjetnog neurona

Kako bi razumjeli što označavaju te težine i kako su matematički formulirani umjetni neuroni prvo se moramo upoznati s radom i građom bioloških neurona. Osnovan građa neurona se sastoji od dendrita, some i aksona. Signali prethodnih neurona se primaju preko dendrita te se nakon njihove transformacije šalju preko aksona idućim stanicama s kojima je povezan svojim završetcima.



Slika 2.1 Osnovna građa biološkog neurona

Isti princip je korišten i prilikom stvaranje umjetnih neurona uz određene izmjene. Izgled jednog takvog neurona ili perceptrona prikazan je Slika 2.2. Nakon primanja ulaznih signala od susjednih perceptrona vrši se njihova sumacija te se dobiveni zbroj transformira putem aktivacijske funkcije. Nakon čega izlazni signal koji putuje aksonom dolazi u interakciju s dendritima susjednog neurona te se množi s težinom sinapse.[2] Težine sinapse su skalarne vrijednosti s kojima se množi svaki ulazni signal x_i, a predstavljaju mijenjajuće parametre koje mreža može sama učiti u procesu treniranja.



Slika 2.2 Matematički zapis umjetnog neurona – perceptor [2]

Ono što nas zanima je kako vektor težina transformira naš vektor ulaznih signala tj. koliki će utjecaj imati pojedini ulaz na naš izlazni signal. Ako pojedini ulaz ne pridonosi dovoljno ispravnoj klasifikacije bit će mu dodijeljen jako mali broj za težinu, kako bi umanjili utjecaj njegovog izlaza. Konačno funkcijom jednog neurona možemo zapisati pomoću jednadžbe:

$$y = \sum težine \cdot ulaz + prag \tag{1}$$

Izlazni signal tako postaje suma umnožaka težina w_i i ulaznih signal x_i na koje smo dodali prag b (eng. *bias*). Budući da je $y \in <-\infty, \infty>$, neuron ne zna treba li se aktivirati ili ne.[2] U tu svrhu uvodimo aktivacijske funkcije. Ako ne koristimo aktivacijske funkcije, svaki sloj neuronske mreže će vršiti linearnu transformaciju, što ograničava naš model i kao takav nije prikladan za kompleksnije zadatke. Uz aktivacijsku funkciju uvodimo još i pojam funkcije gubitka s kojim izražavamo koliko je naš izlazni rezultati udaljen od željenog tijekom faze treniranja na poznatom skupu podataka.

2.2. Aktivacijske funkcije

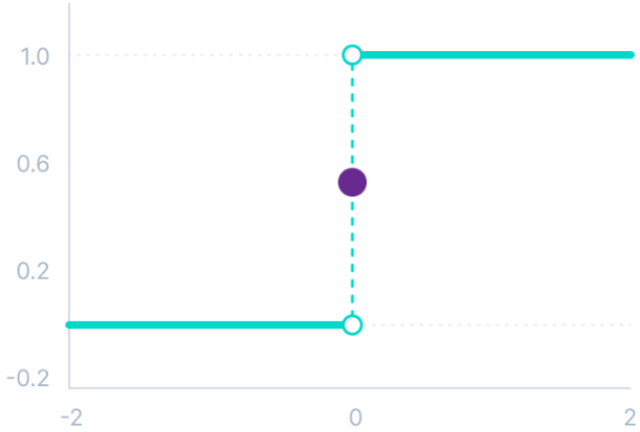
Aktivacijske funkcije dijelimo u tri skupine:

1. Koračna (*Step*) funkcija

2. Linearna funkcija
3. Nelinearne funkcije

Koračna (*Step*) funkcija

Odlučuje aktivnost neurona s obzirom na neku zadanu vrijednost. Ukoliko je ulaz koji dovodimo na aktivacijsku funkciju veći od zadane vrijednosti neuron je aktivan u suprotnom neuron postaje neaktivan.



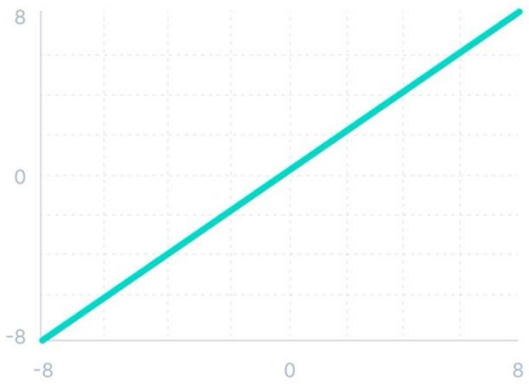
Slika 2.3 Graf koračne funkcije [3]

Matematički zapis takve funkcije je dan izrazom:

$$f(x) = \begin{cases} 0\,, za\ x < 0 \\ 1\,, za\ x \geq 0 \end{cases} \quad \textbf{(2)}$$

Jedan od glavnih nedostataka je nemogućnost korištenja funkcije za izradu klasifikatora u problemima s više klasa. Nadalje, iznos samog gradijenta funkcije iznosi nula, što nam stvara kasnije probleme kod korištenja algoritma propagacije unatrag.[3]

Linearna funkcija



Slika 2.4 Linearna aktivacijska funkcija [3]

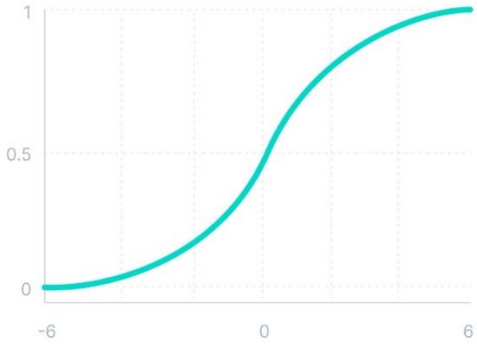
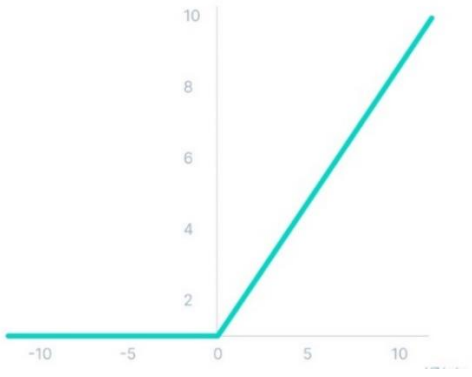
Kod ove funkcije aktivacija je proporcionalna ulaznom signalu. Dobivena suma umnoška vektora težina W i ulaza x_i biva netaknuta. Glavni problem ove funkcije je taj što joj je gradijent konstanta te se ne može primijeniti algoritam propagacije unatrag koji je esencijalan za učenje mreže. Drugi problem je taj što bez obzira na broj slojeva u mreži izlazni sloj će i dalje biti linearan funkcija prvog, što cijelu mrežu pretvara u običnu linearnu regeresiju.[3] Matematički zapis ove funkcije dan je izrazom:

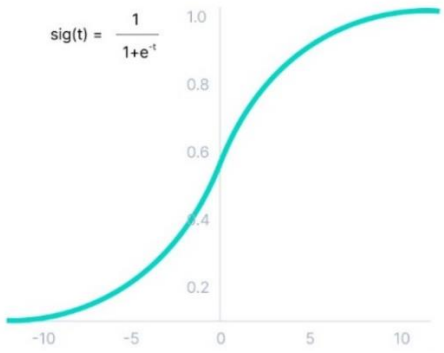
$$f(x, W) = W^T \cdot x + b \tag{3}$$

Nelinearne funkcije

Zbog navedenih problema kod linearne funkcije, u nastavku ćemo se upoznati s nekim nelinearnim funkcijama koje nude rješenja na dane probleme. Budući da postoji velik broj takvih funkcija, one koje ćemo razmotriti u nastavku su sigmoidna funkcija, ReLU funkcija i Softmax funkcija. Pregled grafova i matematičkih zapisa funkcija dan je **Error! Reference source not found.**.

Table 2.1 Nelinearne aktivacijske funkcije

Sigmoidalna funkcija		$f(x) = \frac{1}{1 + e^{-x}}$
ReLU funkcija		$f(x) = \max(0, x)$

Softmax funkcija		$f(z_i) = \frac{e^{z_i}}{\sum_i^K e^{z_i}}$

Sigmoidalna ili logistička funkcija uzima bilo koju dobivenu realnu vrijednost ulaza i kompresira ga u svoju kodomenu koja se kreće u intervalu [0,1]. Zbog tog je prigodna za upotrebu u modelima gdje moramo dobiti vjerojatnost kao izlaz. Iako je funkcija derivabilna, javlja se problem zasićenja gradijenta. Kod jako velikih ili malih ulaza vrijednosti mogu ostati konstante što rezultira jako malim gradijentom. Mali gradijent može uzrokovati nedostatak promjene kod algoritma gradijentnog pada i učenje naše mreže. [4]

Aktivacijska funkcija ispravljanje linearne jedinice ili ReLU (eng. *Rectified Linear Unit*) je funkcija koja za pozitivnu vrijednost ulaza daje istu tu vrijednost, a za negativan ulaz vraća nulu. Budući da u sebi ne sadrži eksponencijalnu funkciju puno brže konvergira i na pozitivnom dijelu funkcije se ne javlja problem zasićenja gradijenta. Iz tih razloga se u praksi pokazala korisna kod veliki neuronskih mreža poput VGGNet, ResNet, MobileNet, itd. .[3] Glavni problem koji se javlja kod ove funkcije je umirući ReLU što znači da će za sve ulaze manje od nule funkcija davati nulu. Kako bi to izbjegli u praksi se uvodi propuštajući ReLU (eng. *Leaky ReLU*). Na sam graf za negativne vrijednosti se uvodi mali nagib koji će davati izlaze proporcionalne ulazu koji su jako blizu nuli. Matematički zapis te funkcije glasi:

$$f(x) = \max(0.01x, x) \tag{4}$$

Softmax funkcija predstavlja generalizaciju sigmoidalne funkcije i ona se uglavnom koristi u zadnjem sloju neuronske mreže kod višeklasnih problema. Za ulaz dobiva nenormalizirani vektor vrijednosti klasa te daje vjerojatnost distribucije svake od klasa.[4]

2.3. Funkcija gubitka

Cilj neuronske mreže je da kroz svoje algoritme prepozna vezu između seta ulaznih podataka i izlaza. Matematički zapis obrade ulaznih podataka i dobivanje predviđenog izlaza možemo

prikazati sljedećom jednadžbom [5]:

$$\hat{y} = \sigma(W^T x + b) \tag{5}$$

Za sam proces treniranja je bitno da poznajemo ulaze i očekivane izlaze. Nakon što pustimo poznate ulaze kroz neuronsku mrežu u procesu zvanom unaprijedna propagacija dobivaju se odgovarajući izlazi. Te izlaze zatim uspoređujemo s ciljanim izlazima, u algoritmu zvanom propagacija unatrag, nakon čeg se mijenjaju početni parametri (težine W^T i $bias\ b$) tako da zadovolje jednadžbu (4).[5] Uloga funkcije gubitka je ta da nam pokaže u kojem smjeru mijenjati vrijednosti početnih parametara da bi se dobiveni izlaz podudarao s očekivanim. Postoji više varijanti funkcija gubitka, a odabir se vrši u odnosu na zadatak koji rješavamo neuronskom mrežom.[4] Funkcije gubitka možemo podijeliti u dvije skupine:

1. Funkcija regresijskog gubitka – koristimo u regresijskim neuronskim mrežama (računanje greške metodom najmanjeg kvadrata)
2. Funkcija klasifikacijsko gubitka – koristimo u konvolucijskim neuronskim mrežama (binarna unakrsna entropija, kategorična unakrsna entropija)

U nastavku ovog rada detaljnije će se obraditi unakrsna entropija jer se će se ona koristiti u kasnijim poglavljima prilikom rješavanja problema klasifikacije.

Unakrsna entropija

Unakrsna entropija (eng. *Cross entropy*) je funkcija koja dolazi iz informacijski znanosti, a u strojnom učenju predstavlja funkciju gubitka koju primjenjujemo na vektor rezultata pod pretpostavkom da su vrijednosti prilagođene na vjerojatnosnu razdiobu.[2][5] Primjenjuje se na problemima višeklasne i binarne klasifikacije.

Za problem binarne klasifikacije koristimo sigmoidalnu funkciju definiranu u prethodnom poglavlju, budući da se njene vrijednosti kreću u skupu [0,1]. Tako definirano binarnu klasifikaciju još zovemo i logističkom regresijom, a definira se:

$$L(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})) \tag{6}$$

U danoj formuli y označava stvarnu klasu, $(1 - \hat{y})$ označava vjerojatnost klase 0, a $\hat{y}$ nam označava vjerojatnost klase 1.[6] Ova kombinacija funkcije gubitka i aktivacijske funkcije se koristi samo u izlaznim slojevima neuronske mreže zbog problema spomenutim u prethodnom poglavlju. Male promijene gradijenta funkcije također onemogućavaju određivanje smjera rasta ili

pada greške što direktno utječe na ažuriranje naših početnih parametara te otežava učenje mreže.[2][4] Kod problema višklasne klasifikacije koristimo unakrsnu entropiju softmax kalasifikatora, gdje je softmax definiran kao i u **Error! Reference source not found.**.. Klasifikator je definiran sljedećom jednadžbom:

$$L = \sum_{i=1}^{K} -y_i \log P(y_i = 1|x) \tag{7}$$

U ovom slučaju K nam iznosi broj klasa, argument pod logaritmom predstavlja vjerojatnost klasifikacije objekta u klasu y_i softmax funkcijom. Ovaj izraz se može iskoristiti i za binarnu klasifikaciju geje je broj klasa onda jednak 2. Nadalje ukupni gubitak ili funkciaj troška se definira kao:

$$J = -\frac{1}{N}\left[\sum_{i=1}^{N} y_i \log P(y_i = 1|x) + (1 - y_i)\log(1 - P(y_i = 1|x))\right] \tag{8}$$

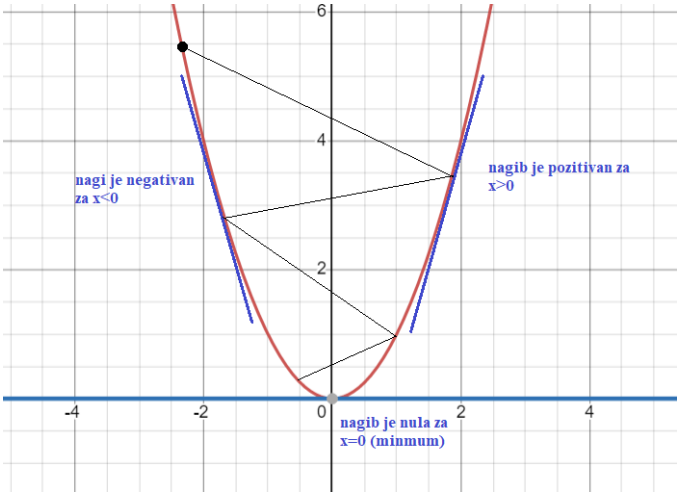
N označava veličinu testnog skupa.

2.4. Algoritmi učenja

Do sada smo objasnili što je jednoj mreži potrebno da bi obradila neki skup podataka te shodno rezultatima djelovala na svoje promjenjive parametre težina i razmaka dane jednadžbom (5). Cilj je minimizirati ukupni gubitak u konačnom broju iteracija, gdje se prilikom svake iteracije postavljaju nove vrijednosti za parametre. Kako bi postigli željene rezultate u našem učenju koristit ćemo se dvama metodama: metoda gradijentnog spusta i metoda propagacije unatrag.[2][7]

Metoda gradijentnog spusta (eng. *Gradient decent*)

Cilj metode gradijentnog spusta je minimizirati funkciju troška (8). Na jednostavnom primjeru kvadratne funkcije možemo vidjeti što nam označava derivacija Slika 2.5, dok će gradijent preuzimat ulogu derivacije kod prostornih funkcija definiranih s više varijabli. Idealna vrijednost gradijenta funkcije troška iznosi nula pa nam je cilj kretat se u smjeru najvećeg pada kako bi što prije dosegli minimum. Krećemo iz točke u ovisnosti o proizvoljno postavljenim parametrima u kojoj računamo gradijent funkcije troška koji će nam pokazat u kojem smjeru se nalazi naš minimum.[7]

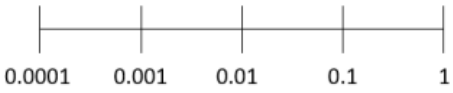


Slika 2.5 Graf kvadratne funkcije

Početni parametri težina W i pomaka b su zadani vektorom Θ. Nakon što smo izvršili linarnu propagaciju računamo vrijednost gradijenta funkcije cijene $\nabla J(\Theta)$ te vršimo izmjenu početnih parametara na sljedeći način:

$$\Theta^{t+1} = \Theta^{t} - \alpha \nabla J(\Theta^{t}) \tag{9}$$

Broj iteracije nam predstavlja t, a α nam predstavlja novi parametar kojeg nazivamo stopom učenja. O njemu nam ovisi konvergiranje parametara u smjeru minimuma, ako je prevelik divergirat ćemo u suprotnom će konvergirati prema minimumu jako malom brzinom. Na početku ga postavljamo na proizvoljna vrijednost koja je dobivena u praksi i iznosi $\alpha = 0.01$.[7] Ukoliko nam je prespora konvergencija iznos ćemo povećati na 0.1 pritom pazeći da nam ne počne divergirati.



Slika 2.6 Skala za koeficijent učenja[7]

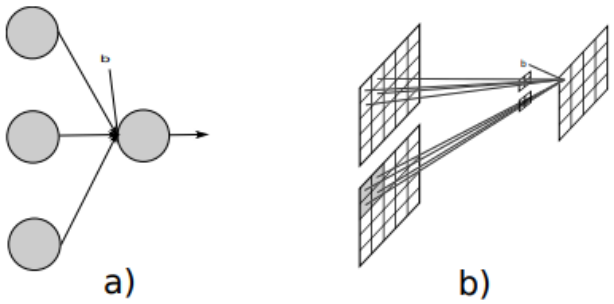
Gradijent troška funkcije možemo dobiti **algoritmom propagacije unatrag.** Primjenom lančanog deriviranja (10) na funkciju troška J u smjeru vektora Θ dobivamo traženi gradijent.

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}. \tag{10}$$

3. Konvolucijske neuronske mreže

Konvolucija je matematička operacija koja se vrši nad dvjema funkcijama, s domenom u realnom u području, u svrhu dobivanja poboljšanje verzije jedne od dviju funkcija. Konvolucijske neuronske mreže su posebna vrsta mreža koja je specijalizirana za rad nad višedimenzionalnim strukturama. Što znači da za ulaz možemo imati sliku nad kojom će se izvršiti konvolucija te kao rezultat dobiti raspoznavanje različitih značajke na njoj.[8][9]

Arhitektura konvolucijskih neuronskih mreža jednaka je kao i kod standardnih neuronskih mreža, a sastoji se od ulaznog sloja, jednog izlaznog i jednog ili više skrivenih slojeva. Ono što je specifično za konvolucijske neuronske mreže je upotreba samih konvolucijskih slojeva i slojeva sažimanja. Opisana struktura neurona u prethodnom poglavlju za izlaz je davala skalara, međutim upotrebom novih slojeva izlazi više nisu jednodimenzionalni već dvodimenzionalni i nazivamo ih mapama značajki (eng. *feature maps*). Sukladno tome više se ne koriste težine već se koriste jezgre (eng. *kernels*).[10][11]



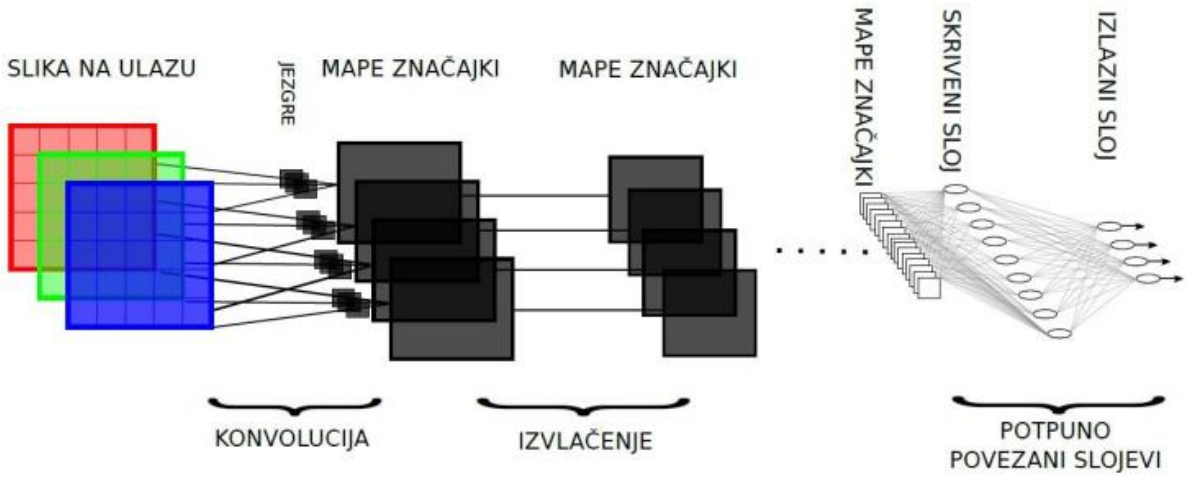
Slika 3.1 a)neuroni i njegova povezanost i b) Mapa značajki i primjer njezine povezanosti

Prvi model CNN mreže *Neocognitron* razvio je Kunihiko Fukushima u 80-im godinama prošlog stoljeća, koji se bazirao na radu Hubbela i Wiesala o načinu slaganja neurona u vizualnom korteksu iz 1959.. *Neocognitron* se sastojao od više slojeva koji su automatski učili hijerarhiju izvlačenja značajki tokom prepoznavanja uzorka. Prvi značajni pomak je došao 1989. sa razvojom LeNet mreže koja je koristila algoritam računanje greške propagacijom unatrag u svrhu učenja parametara.[11]

Kovolucijske mreže se koriste i kod nadziranog i nenadziranog učenja. Kod nadzirano učenja ulazni i izlazni podaci su nam poznati te mreža uči preslikavanje između njih. Nenadzirano učenje podrazumijeva da nisu poznati izlazni podaci za dani skup ulaznih podataka već model pokušava procijeniti moguću distribuciju zadani ulaznih podataka.[10][11]

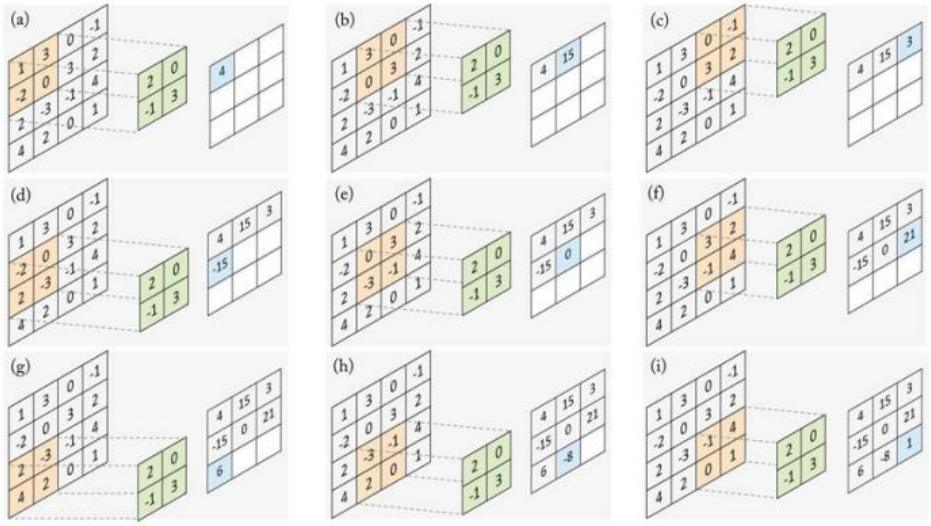
3.1. Konvolucijski sloj neuronske mreže

Opća struktura konvolucijske neuronske mreže dana Slika 3.2 na čiji ulaz se dovodi slika u boji s 3 kanala (RGB). Nakon tog slijede konvolucijski slojevi u kojima ulazni podaci konvolviraju sa zadanom jezgrom.



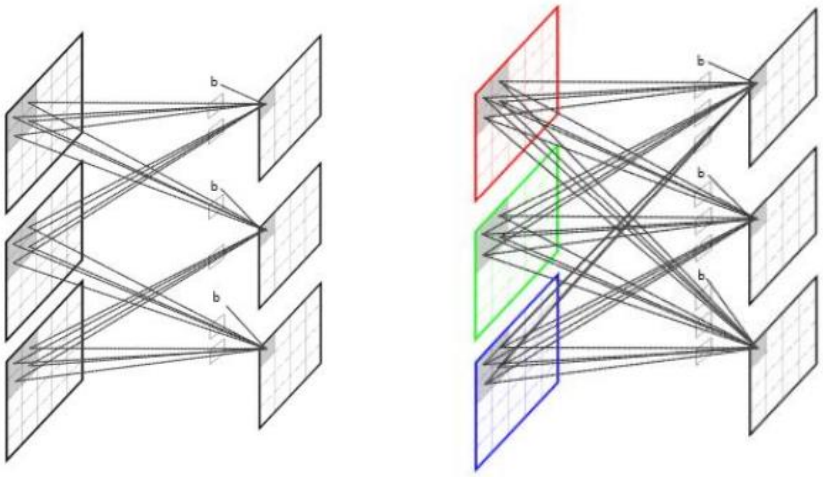
Slika 3.2 Osnovana struktura konvolucijske mreže [10]

U nastavku ćemo razmotriti 2D konvoluciju kako bi detaljnije pokazali kako konvolucijski sloj transformira naše podatke. Ako su nam ulazne mape značajki dane u formi matrice 4x4 dimenzija i pripadajuća jezgra u formi matrice 2x2, konvolucijski sloj će množiti jezgru sa vrijednostima matrice nad kojima se nalazi(Slika 3.3) te tako generirati izlazne vrijednosti jednu po jednu. Jezgra se tokom cijelog procesa pomiče za korak po dužini ili širine mape značajki. U ovom slučaju je riječ o koraku vrijednosti 1 piksel. Iznos koraka se može mijenjati.



Slika 3.3 Konvolucija[11]

Povezanost pojedinih kovolucijskih slojeva reprezentira se jezgrom što dovodi do zaključka da svaka mapa značajki mora imati onaj broj jezgri koliko je imala i mapa značajki iz prethodnog sloja s kojim je povezana. Međutim svi slojevi ne moraju biti povezani moguće je koristiti i raspršenu povezanost. [10][11]



Slika 3.4 Lijevo je prikazana raspršena povezanost, a desno potpuna povezanost [10]

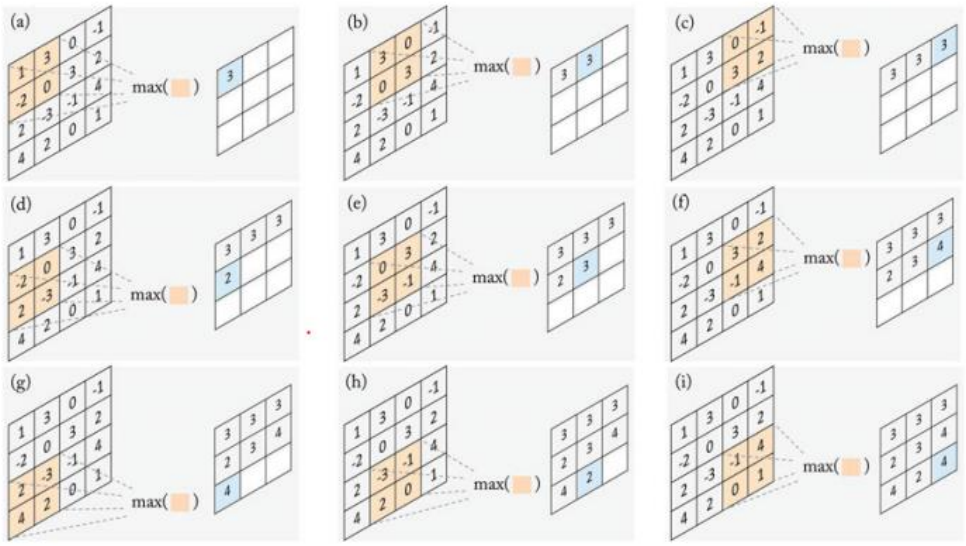
3.2. Sloj sažimanja

Konvolucijski sloj slijedi sloj sažimanja koji u konačnici ima istu funkciju, a to je da reducira veličinu mape značajki koja je konvolvirala u prethodnom koraku. Smanjivanjem konvolvirane mape značajki smanjiva se memorijski zahtjevi za implementacije neuronske mreže. Također utječemo i na učinkovitost treniranja izdvajanjem dominantnih značajke koje su prostorno i rotacijski nepromjenjive.[12]

Razlikujemo dva tipa sažimanja: sažimanje maksimalnom vrijednošću (*Max Pooling*) i sažimanje usrednjavanjem (*Average Pooling*). Slično kao i kod konvolucije moramo definirati veličinu jezgre sažimanja te njezin korak. Prilikom treba pripaziti na njihov izbor kako ne bi došlo do gubitka informacija ukoliko su vrijednosti dimenzija jezgre ili koraka prevelike. Kod sažimanja maksimalnom vrijednošću dobivamo maksimalnu vrijednost mape značajki koju pokriva jezgra na našoj mapi značajki Slika 3.5.. Sažimanje uprosječavanjem vrijednosti funkcionira na jednak način s tim da za izlaznu informaciju daje prosječnu vrijednost u području na kojem se jezgra nalazi.[12]

U kombinacija sa konvolucijskim slojem dobivamo i-ti sloj konvolucijske neuronske mreže. Broj takvih slojeva u mreži će ovisiti o kompleksnosti našeg zadatka no povećavanje slojeva će također

rasti i zahtjev za većom memorijom. Pozitivna strana povećanjem broja slojeva je ta što će se moći zabilježiti detalji niskog stupnja prepoznavanja.



Slika 3.5 ***Max pooling***

3.3. Potpuno povezani slojevi

Potpuno povezani slojevi (eng. *Fully connected layers*) se nalaze na samom kraju neuronske mreže. U potpunosti su povezani sa svakim neuronom iz prethodnog sloja te njeg djeluje jezgrom veličine 1x1. Možemo postaviti i više takvih slojeva za redom. Rad zadnjeg sloja možemo prikazati kao djelovanje nelinearne funkcije nad jednadžbom (3). Nelinearna funkcija koja se najčešće primjenjuje u ovom sloju je *SoftMax* funkcija objašnjena u prethodnim poglavljima.[12] Ukoliko se radi o binarnoj klasifikaciji izlaz će biti samo jedan broj, a ako izvršavamo problem klasifikacije u *K* broj klas tad izlaz sadrži vektor vrijednosti za svaku klasu.

3.4. Funkcija gubitka

Posljednji sloj u konvolucijskoj neuronskoj mreži služi samo u procesu treniranja. Korištenjem funkcije gubitka u ovom sloju želimo dobiti odnos dobivene izlazne vrijednosti i ciljanje izlazne vrijednosti. Tip funkcije gubitka ovisi o zadanom krajnjem problemu. Najčešći mogući slučajevi za konvolucijske mreže su [11]:

1. Binarna klasifikacija
2. Identifikacija verifikacijom
3. Višeklasno razvrstavanje

4. Regresija

Najčešće korištena funkcjia gubitka, koj će se koristiti i u nastavku rada, je gubitak unakrsne entropije detaljnije opisana u prethodnim poglavljiva. Neke od funkcija koje se koriste su: SVM *hinge loss*, kontrasnni gubitak, gubitak očekivanja, euklidski gubitak.

3.5. Inicijalizacija težina

Za stabilno treniranje naše konvolucijske neuronske mreže potrebno je pravilno postaviti početne vrijednosti za parametre težina. Postavljanjem na jako male ili velike vrijednosti mogu nam se pojaviti problemi kod gradijentom prilikom propagacije unatrag do početnih slojeva. Ako je vrijednost premala, postoji opasnost od pojave nestajućeg gradijenta. U suprotnom, javlja se problem eksplodirajuće gradijenta gdje se javlja velika težina ažuriranja koja premašuje minimalnu vrijednost. U nastavku ćemo objasniti par različitih pristupa.[12][13]

Gaussovom inicijalizacijom postavljamo matricu s nasumičnim vrijednostima dobivenim iz normalne distribucije s očekivanom vrijednošću 0 i malom standardnom devijacijom. [13]

Uniformna nasumična inicijalizacija koristi uniformnu raspodjelu za dobivanje matrice nasumičnih vrijednosti te generalno nema velike razlike u odnosu na gaussovu. Međutim za obe se javlja problem pojave eksplodirajućeg i nestajućeg gradijenta prilikom treniranja dubokih neuronskih mreža. [13]

Postoji i mogućnost korištenja postupka podešavanja. Mrežu s nepoznatim težinama treniramo na većoj bazi podataka koja je slična našom početnom. Potom naučene težine spremimo i iskoristimo na novom skupu podataka. Znanje naučeno na jednom skupu prenosimo na drugi.[11]

Xavier inicijalizaciju koristimo kako bi riješili problem proporcionalnosti varijance izlaznih i ulaznih veza. Definiramo novu varijancu koja je ovisna o broju ulaznih i izlaznih veza neurona:

$$Var(w) = \frac{2}{n_{f-in} + n_{f-out}} \tag{11}$$

S tako izračunatom varijancom inicijaliziramo vrijednosti težina w. Računanjem težina na ovaj način moramo uzeti u obzir pretpostavku postojanja linearne veze između ulaza i izlaza. U praksi neki neuroni posjeduju ReLU nelinearnost te ne slijede pretpostavke *Xavier* inicijalizacije, stoga uvodimo novi izraz za ReLU *aware scaled* inicijalizacija:

$$Var(w) = \frac{2}{n_{f-in}} \tag{12}$$

Postižemo bolju statističku točnost nego kod prethodne incijalizacije.[11]-[14]

3.6. Metode regulacije

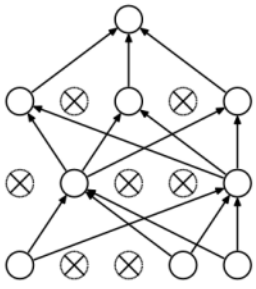
Kod dubokih neuronskih mreža tijekom procesa učenja ,zbog velikog broja parametara, model ima dobre rezultate na testnoj bazi podataka dok kod novih podataka ima loš performans. Cilj regulacije je izbjeći ovaj problem koristeći se regulacijskim idejama podijeljenim u sljedeće kategorije:

1. reguliraju mrežu korištenjem tehnika razine podataka - *data agumentation* metode;
2. unose stohastičko ponašanje u neuronske aktivacije – *dropout* metoda;
3. normaliziraju statistiku skupa u aktivaciji značajki - *batch normalization*;
4. koriste "spajanje sličnih ili različitih elemenata u zajednicu" (engl. decision level fusion) radi izbjegavanja prekomjerne specijalizacije modela - *ensemble model averaging*;
5. unose ograničenja težinskih vrijednosti mreže;
6. se vode validacijskim setom radi zaustavljanja procesa učenja – *early stoping*.[11]

Budući da se prve tri metode koriste u nastavku ovog rade, navest će se prošireno objašnjenje svake.

Data agumentation ili agumentcija skupa za učenje proširiva već postojeći skup stvarajući kopije podataka nad kojima je izvršena translacija, rotacija, promjena svjetlina i sl. Zahvaljujući invarijentnosti konvolucijskih neuronskih mreža možemo u proces učenja primijenit sintetički skup podataka u svrhu povećana baze i boljeg treniranja [11][12]

Dropout regularizacija je jedna od poznatijih tehnika regulacije. Svakom se neuronu dodjeljiva fiksna vrijednost koja označava vjerojatnost aktivacije pojedinačnog neurona. Ovakav način regularizacije ulaza ima stimulirajući efekt na učenje neuronske mreže jer pri svakoj iteracije uči novu konfiguraciju neurona.



Slika 3.6 Neuronska mreža nakon upotreba dropout-a

Batch normalisation ili normalizacija serije podataka je metoda koja se u praksi pokazala idealna za ubrzavanje i stabilizaciju treniranja umjetnih neuronskih mreža tako što vrši normalizacija ulaznih podataka u slojeve ponovnim skaliranjem i centriranjem.

Promjena unutarnje kovarijance se odnosi na promjenu distribucije aktivacija kroz svaki sloj tijekom ažuriranja parametara tijekom treniranja. Budući da skriveni slojevi konvolucijske mreže žele modelirati tu distribuciju koja se konstantno mijenja (visoka vrijednost unutarnje kovarijance), proces treniranja će sporo konvergirati. Primjenom normalizacije distribucije podataka rezultira konstantom aktivacijskom distribucijom tijekom procesa treniranja što vodi do brže konvergencije i stabilnijeg procesa. Primjenjuje se prije korištenja aktivacijske funkcije.[12]-[15]

3.7. Metode optimizacije

Proces treniranja neuronske mreže je jako osjetljiv na inicijalizaciju ulaznih podataka i postavljanje stope učenja (eng. *learning rate*). Potencijalni probleme na koje nailazi naša neuronska mreža i koje je potrebno zaobići se pojavljuju u obliku nestajućeg i eksplozivnog gradijenta, zaustavljanja u lokalnom minimumu te pojava sedlaste točke u kojoj iznos parcijalnih derivacija iznosi nula. Kako bi izbjegli ove potencijalne probleme algoritma gradijentnog spusta uvodimo potrebne optimizacije.[12]

Optimizacija momentom nije ništa drugo već poboljšana verzija stohastičkog gradijentnog pada. Moment možemo povezati s momentom kretanja mase koji predstavlja umnožak brzine tijela i mase koju tijelo ima. Za potrebe neuronskih mreža masa se postavlja na jediničnu vrijednost stoga nam moment označava brzinu gibanja. Moment je faktor koji će djelovat na ubrzanje ili usporavanje gradijenta u ovisnosti o njegovom položaju (jednak smijer ubrzavanje, suprotan smijer usporavanje). Tipična vrijednost momenta u kod stohastičkog gradijentnog pada iznosi m=0.9.[12]

Upotrebom Nesterovog momenta usporavamo našu točku dok se približava minimumu na način da računa gradijent u idućoj aproksimiranoj točki umjesto u trenutnoj. Kao rezultat naš algoritam može unaprijed vidjeti kad se zaustaviti bez da zapne u lokalnom minimumu.[12]

AdaGrad algoritam se koristi kad je potrebno ažuriranje svakog parametra prema njegovoj frekvenciji u skupu za treniranje. Ovaj proces se ponavlja prilikom svake iteracije. Stopa učenja svakog parametra se dijeli s akumuliranim kvadratom prijašnjih gradijenata te tako izbjegavamo potrebu za ručnim postavljanjem iste. Ova metoda se pokazala posebno dobrom kod raspršenih gradijenata.[12]

ADAM (eng. *ADAptive Moment Estimation*) procjenjuje stopu učenja za svaki parametar računanjem prvog i drugog momenta gradijenta te na osnovu toga vrši njeno ažuriranje. Kod prethodne metode postoji opasnost od prestanka mijenjanja parametara, zbog niske stope učenja,

što smo ovom metodom izbjegli. ADAM se u praksi pokazao kao izbor za probleme računalnog vida temeljene na dubokom učenju, zbog efikasne konvergencije nad problemima većih razmjera.[12]

4. VGG arhitekture neuronske mreže

Karen Simonyan i Andrew Zisserman su 2014. u svom rad predstavili novu strukturu neuronske mreže koja se temelji na VGG (*Visual Geomtry Group*) blokovima.[16] U svom modelu su po prvi put prikazali povećanje dubine neuronske mreže koristeći male konvolucijske filtere dimenzija 3x3. Dokazali su da korištenjem malih filtera u više konvolucijskih slojeva može simulirati receptivno ponašanje polja filtera veličine 5x5 ili 7x7, dok se broj parametara smanjio.

Ova saznanja su iskoristili kao bazu za ILSVRC 2014 natjecanje gdje su osvojili drugo mjesto u klasifikaciji i prvo u lokalizaciji objekta.[16]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Slika 4.1 VGG struktura neuronskih mreža [16]

Na *ImageNET* bazi podataka koja se sastoji do 14 milijuna slika raspoređeni u 1000 klasa postigli su točnost od 97.2%. korištenjem neuronske mreže sa 16 konvolucijskih slojeva. Dodatna specifičnost njihove strukture je što nisu koristili sloj sažimanja nakon svakog sloja.[17]

5. Analiza rada VGG arhitekutre neuronske mreže

5.1. Priprema platforme za rad

U svrhu ovog rada koristit će se više online alata zbog potrebe za velikom memorijom .

Google drive je besplatna *cloud* platforma za pohranu i dijeljenje datoteka u *online* obliku kojem možete pristupiti upotrebom internetskih pretraživača.

Google Colab je online platforma koja omogućava, kroz internetski preglednik , programiranje proizvoljnog koda u Python-u bez upotrebe lokalnih resursa (procesora, memorije, itd.). Posebno je prilagođen za probleme strojnog učenja i analize velikih baza podataka. *Colab* je podržan *Jupyter-om* softver otvorenog koda koji ima već postavljeno sučelje omogućavajući besplatan pristup TPU i GPU karticama.[18] Glavna prednost korištenja grafičke procesorske jedinice u problemu umjetnih neuronskih mreža je ta što omogućava vršenje na tisuće operacija u jednom ciklusu. Shodno tom proces treniranja postaje puno kraći čak i na velikim bazama podataka u koju spada i Kaggle-ova baza podataka slika mačaka i pasa koje se koristi u ovom radu.

5.2. Python

U prethodnom poglavlju je spomenuto da će se koristit Python kao programski jezik za razvoj naše strukture neuronske mreže s naglaskom biblioteku *TenserFlow* i okvir dubokog učenja *Keras*.

Keras je otvoreni kod koji nam omogućava jednostavno sučelje/okvir dubokog učenja u Pythonu u kojem se jednostavno može definirat i istrenirati bilo koji model dubokog učenja.[20] Glavne značajke ovog okvira su:

- Brzo pokretanja koda na TPU,GPU ili CPU
- Podržava konvolucijske mreže za računalni vid i ponavljajuće mreže za obradu sekvenci
- Visoka zastupljenost u industriji
- Jednosatvan korisnički API s povratnom informacijom koji omogućava brzo prototipiranje modela dubokog učenja.[20]

TenserFlow je besplatna biblioteka za strojno učenje koja se koristi grafovima protoka za numeričko učenje. Rubovi grafova predstavljaju polja s n dimenzija koje još zovemo i tenzorima, a položena su među čvorovima koji predstavljaju matematičke operacije. Podržava veliki broj aplikacija s fokusom na treniranju dubokih neuronskih mreža i provođenje strojnog učenja. Sadrži i biblioteku *TenserBoard* kako bi pratili bitne metričke vrijednosti tokom treniranja poput točnosti i gubitaka. [21]

5.3. Organizacija baze podataka

5.3.1. Opis problema

Za potrebe ovog rada koristila se slobodno dostupna baza podataka dobivena od strane Kagglea. Baza podataka slika mačaka i pasa je dio veće baze podataka nastale od strane *Microsofta* i *Petfinder* , koja se koristila za CAPTCHA (eng. *Completely Automated Public Turing*) test koji imao svrhu raspoznavanja stroja od ljudi.[22]

Na izdvojenoj bazi podataka od 25000 slika potrebno je razviti konvolucijsku mrežu koja će raspoznavati pse od mačaka. Zatim na osnovnom modelu uvesti poboljšanja koje treba usporediti s rezultati dobivenim iz osnovnog modela. Konačno usporediti performans s unaprijed testiranom arhitekturom neuronske mreže VGG-16.

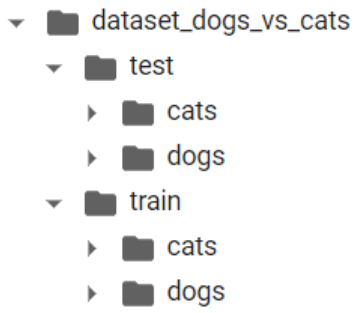
5.3.2. Priprema baze podataka

U bazi se nalazi jednak broj slika mačaka i pasa koje su imenovane odgovarajućom oznakom :

```
cat.0.jpg
...
cat.124999.jpg
dog.0.jpg
dog.124999.jpg
```

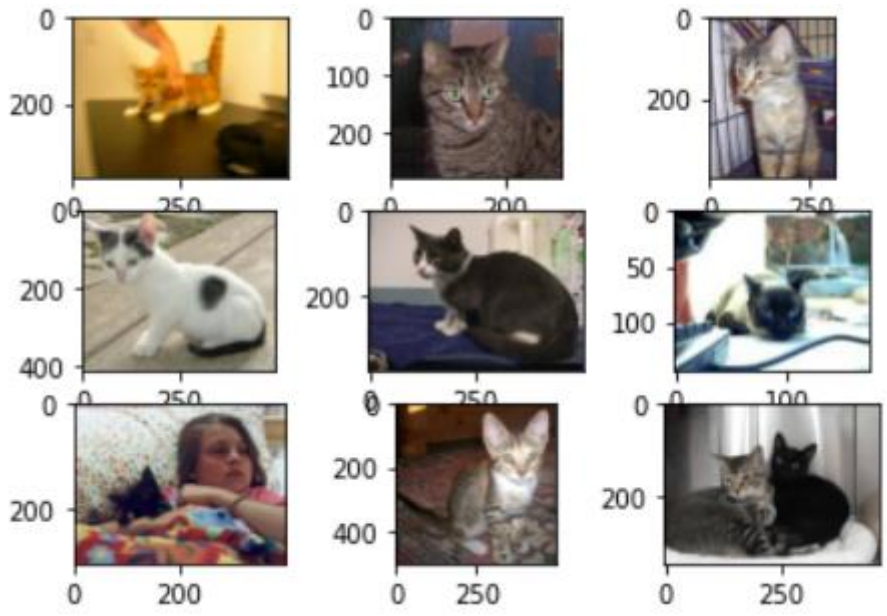
Slika 5.1 Imenovanje slika

One se trenutno nalaze sve u istom direktoriju te za potrebe neuronske mreže ih je potrebno razvrstati u strukturu danu Slika 5.2.



Slika 5.2 Struktura direktorija

Budući da slike u danoj bazi dolaze u raznim veličinama, a iz prethodnih poglavlja znamo da ulazni podaci moraju biti jednake veličine, sve slike ćemo svesti na jednaku dimenziju od 200x200 piksela.



Slika 5.3 Nasumične slike mačaka izvučene iz baze prije obrade

Postoji mnogo načina na koji možemo promijeniti veličinu, koji će ovisiti o dostupnosti memorije koju imamo. U nastavku rad za promjenu veličine slike će se koristiti klasa *ImageDataGenerator* i API *flow_from_directory()*. Izvedba je nešto sporija, ali zahtijeva manje memorije. Detalji klase će biti objašnjeni idućem poglavlju.[23]

5.4. Osnovni model VGG arhitektur neuronske mreže

U ovom poglavlju ćemo pokazati razvoj osnovnog modela s minimalnom performansom kao podlogu za daljnja poboljšanja. Koristit ćemo VGG strukturu slaganja blokova, a svaki blok se sastoji od konvolucijskog sloja s malim filterom veličine 3x3 kojeg slijedi sloj sažimanja koji koristi princip maksimalne vrijednosti. S dubinom mreže broj filtera u svakom bloku će pratit slijed povećanja 32, 64, 128, 256.[23] Kako bi osigurali podudaranje veličina mapa kalsifikacija s ulazima koristi se *padding*. *Padding* dodaje sloj nula kako bi veličina mapa filtera bila jednaka ulazu.

Svaki blok koristi ReLU aktivacijsku funkciju i inicijaliziaciju težina zvanu He_uniform. Incijalizacija težina uzima uzorke Gaussovom distribucijom iz skupa [-limit, limit] :

$$limit(w) = \frac{2}{n_{f-in}} \quad \textbf{(13)}$$

Gdje se izraz podudara s ReLU *aware scaled* incijalizacijem definirnaom u prethodnim poglavljima.

Definiranje blokova glasi:

```
# blok 1
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',input_shape=(200, 200, 3)))
  model.add(MaxPooling2D((2, 2)))
# blok 2
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(MaxPooling2D((2, 2)))
# blok 3
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(MaxPooling2D((2, 2)))
```

Slika 5.4 Definirani blokovi VGG strukture

Model ćemo optimirati sa stohastičkim gradijentnim padom s korakom učenja 0.001 i momentom 0.9. Početne vrijednosti su dobivene u praksi kao dobra polazna točka koju možemo kasnije prilagođavati.

Potpuni model jednog bloka glasi definirat ćemo u funkciji *define_model()* :

```
# definiranje pomoću jednog bloka VGG strukture
def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = SGD(learning_rate=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

Slika 5.5 konvolucijsak neuronska mreža definarana jednim VGG blokom

Vidimo da se na konvolucijski sloj dodao potpuno povezani sloj. Kako bi imali pravilan ulaz u našu mrežu slike koje smo razvrstali u direktorije moramo izmijeniti sve na istu veličinu te iznose piksela skalirati na interval između [0,1]. Kako je zaključeno u prethodnim poglavljima skaliranje piksela se vrši kako bi vrijednosti gradijenta ostale konzistentne. Prevelike vrijednosti gradijenta usporavaju trenirnaje mreže.

Korištenjem *flow_from_directory()* API-ja na generatoru podataka definiranom kao na Slika 5.6. Potrebni su nam još iteratori za oba skupa podataka u kojima ćemo dodatno definirati neke od argumenata :

- *class_mode* – naznačimo da se radi o binarnoj klasifikaciji
- *batch*_size - veličinu serije 64
- *target_size* - ciljanu dimenziju slike 200x200

```
# definiran data generator
datagen = ImageDataGenerator(rescale=1.0/255.0)

# priprema iteratora
train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
  class_mode='binary', batch_size=64, target_size=(200, 200))
test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
  class_mode='binary', batch_size=64, target_size=(200, 200))
```

Slika 5.6 Priprema ulaznih podataka

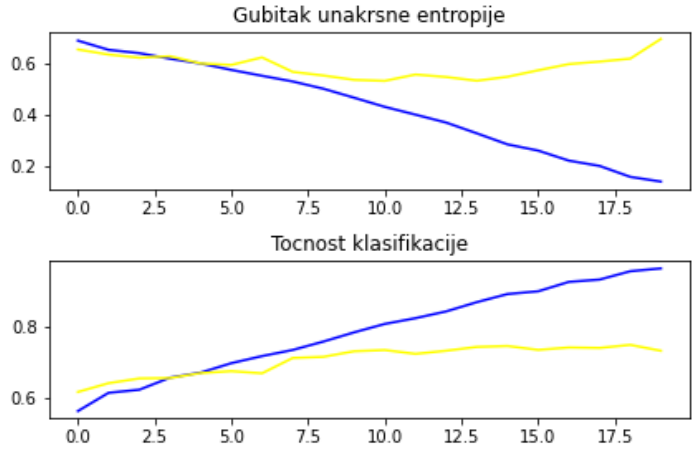
Za kraj nam je samo ostalo istrenirati našu mrežu na zadanoj bazi podataka za treniranje. Nakon treniranja konačni model možemo ispitati na testnoj bazi podataka i procijeniti točnost klasifikacije.

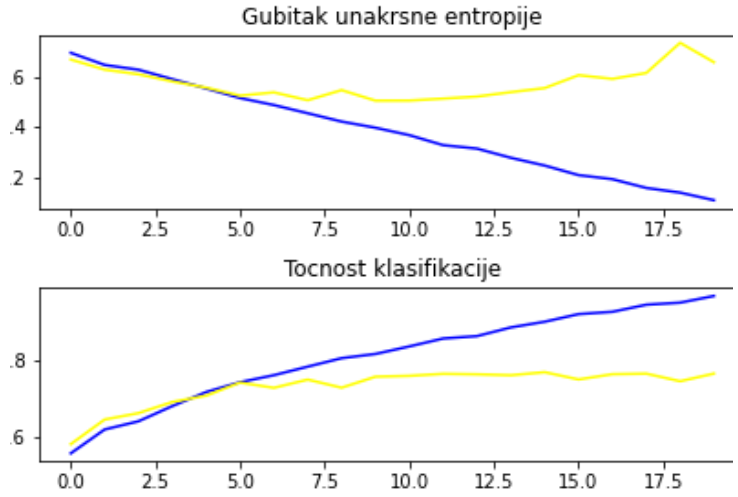
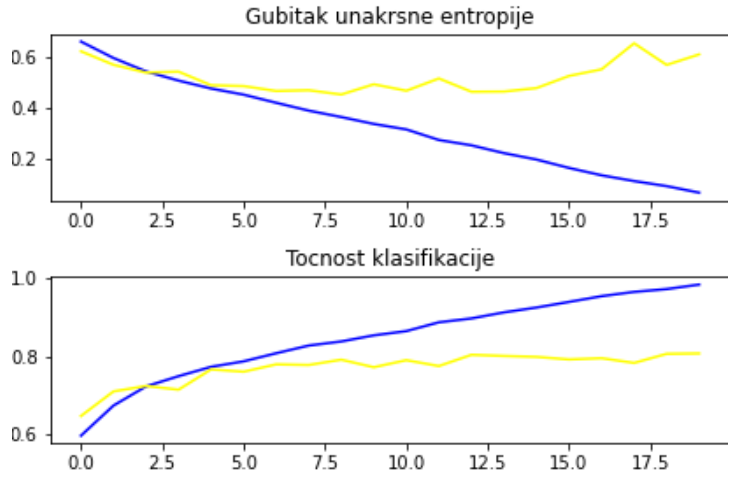
```
# fit model
history = model.fit(train_it, steps_per_epoch=len(train_it),
  validation_data=test_it, validation_steps=len(test_it), epochs=20, verbose=0)

# procjena modela
_, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
print('> %.3f' % (acc * 100.0))

# krivulja učenja
summarize_diagnostics(history)
```

Slika 5.7 Treniranje i procjnea točnsoti kalsifikacije

Izabran je mali broj epoha kako bi se ustanovilo uči li naša mreža. U nastavku ćemo u tablici prikazati razlike u korištenju jednog, dva i tri VGG bloka u arhitekturi neuronske mreže.

Table 5.1 Uspoređivanje rezultata

1 VGG BLOK		Optimizacija: SGD Točnost: 73.108 %

2 VGG BLOKA		Optimizacija: SGD Točnost: 76.519 %
3 VGG BLOKA		Optimizacija: SGD Točnost: 80.739%

S obzirom na dosegnute točnosti u različitim modelima možemo zaključiti da povećanjem VGG blokova raste i točnost klasifikacije. Treniranoj bazi podataka odgovara plava linija, a žuta odgovara testnoj bazi podataka. U sva tri slučaja možemo primijetit da nakon određenog broja epoha se javlja *overfitting* što znači da model ima lošu klasifikaciju na testnoj bazi podataka. S povećanjem kapaciteta *overfitting* se javlja sve ranije.

Rezultati sugestiraju da bi trebalo primijeniti metode regularizacije kao što su *data agumentation, dropout* metoda, normalizacija serije itd.

5.4.1. Uvođenje poboljšanja

Osnovni model koji će se uzeti je model s tri VGG bloka jer je on pokazao najveću točnost u prethodnom poglalvju. Poboljšanja osnovnog modela će se vršiti kroz metode regularizacije i metode optimizacije te njihovom kombinacijom .

Prve će se primijeniti metoda optimizacije ADAM funkcijom. Parametri koje sadrži ADAM funkcija su:

- alpha – označava stopu učenja ili veličinu koraka
- beta1 – eksponencijalna stopa raspadanja procijenjena za prvi momenta
- beta2 – eksponencijalna stopa raspadanja procijenjena za drugi momenta; kod problema raspršenih gradijenata ova vrijednost se postavlja jako blizu jedinice
- epsilon – jako mali broj koji sprječava dijeljenje s nulom u implementaciji. [24]

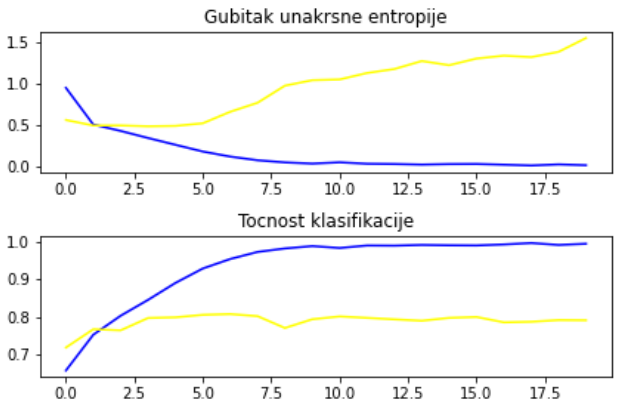
Početne vrijednosti argumenata funkcije su uzete prema [24], a optimizator se definira na sljedeći način:

```
# compile model
opt =Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

Slika 5.8 Definiranje ADAM optimizatora

Dodatni parametar koji smo mijenjali broj epoha. U ovom slučaju smo radili s 20, a rezultati su prikazani u tablici.

Table 5.2 Poboljšavanjem uvođenja ADAM funkcije

3 VGG BLOK		Optimizacija: ADAM Točnost: 78, 375% Epohe: 20

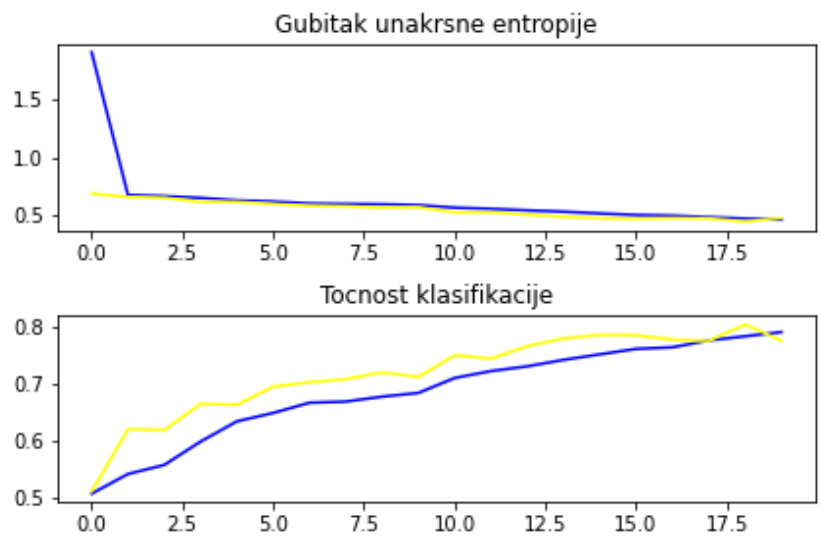
Iako je postignuto velika točnost klasifikacije na treniranom setu mreža je postala ne osjetljiva na testne podatke i javlja se problem ekspandirajućeg gradijenta. Zbog toga na taj model ćemo primijeniti metode regularizacije *Dropout* metodom i *Data agumentation metodu* objašnjene u poglavlju 3. Mala količina *Dropouta* će biti primijenjena nakon prvog VGG bloka, a za svaki sljedeći ćemo je povećati za 0.1. U nastavku će biti prikazana define_model() funkcija u kojoj je prikazana definicija *Dropouta* slika.

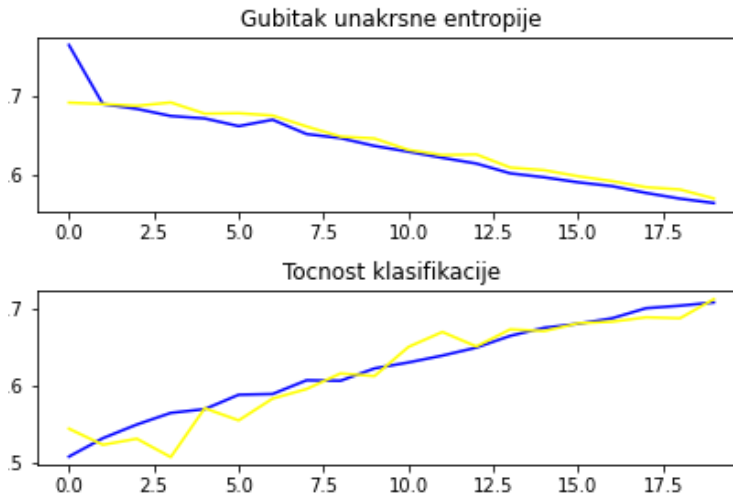
```
def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',padding='same',input_shape=(200, 200, 3)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.2))
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.3))
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.4))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dropout(0.5))
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
  model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

Slika 5.9 Definira funkcija neuronske mreže s poboljšanjima

U svrhu usporedbe regularizaciju ćemo primijeniti i na SGD i ADAM funkciju optimizacije. Rezultati su dani sljedećom tablicom:

Table 5.3 Poboljšanje *Dropout metodom*

3 VGG BLOK		Optimizacija: ADAM Točnost: 85.277%

3 VGG BLOKA		Optimizacija: SGD Točnost: 71.315 %

U prvom slučaja kad koristimo ADAM funkciju postigli smo veliku točnost klasifikacije s obzirom na broj blokova u arhitekturi. Krivulja gubitka za trenirani skup ima umjeren velik gubitak u početku dok se prilikom treniranja smanjiva te u konačnici se izravni što nam govori da povećanje baze trenirnaog skupa nće povećati točnost klasifikacije. Ista stvar vrijedi iza trenirani set podataka.

U drugom slučaju koristimo SGD funkciju optimizacije i analizirajući dobivene grafove vidimo da se pretreniranje (eng. *overfitting*) smanjio te daljnjim povećavanjem brojem epoha bi se mogla postići veća klasifikacijska točnost. Moguća promijena *dropout* faktora bi mogla pridonijeti dodatnom poboljšanju.

Data agumentation

Na osnovnu arhitekturu VGG mreže u nastavku ćemo primijeniti *Data agumentation* metodu. Korištenjem ove metode umjetno proširujemo bazu podataka za treniranje tako što vršimo stvaranje modificiranih verzija slika koje se već nalaze u bazi podataka. Treniranjem na većoj bazi podataka može rezultirati u preciznijim modelima.

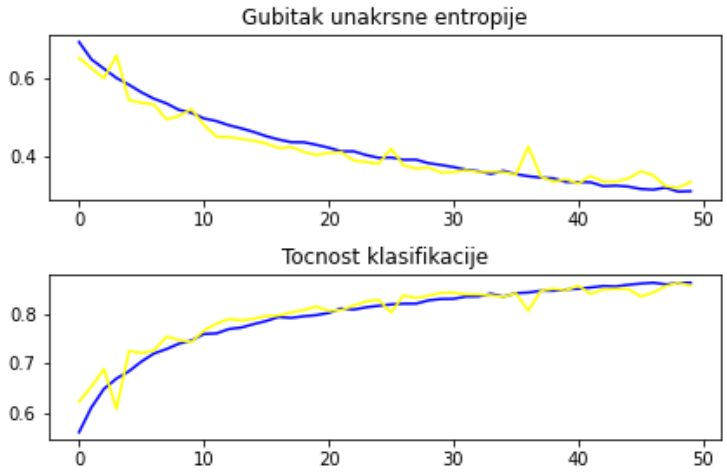
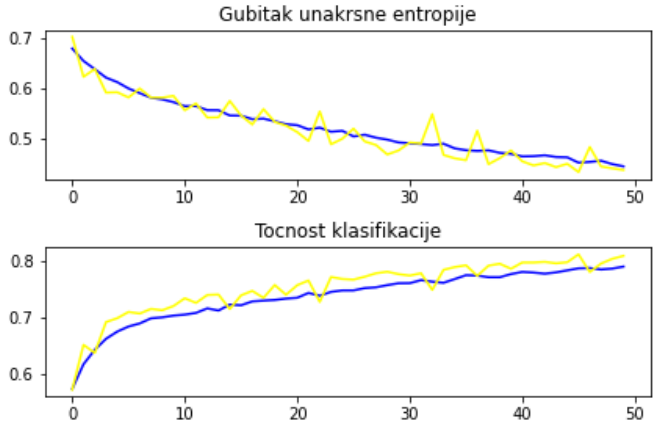
Ova metoda zahtjeva da imamo dva različita *ImageDataGenerator* za trenirani i testni skup podataka te na osnovu toga kreiramo i odgovarajuće iteratore Slika 5.10. Slike iz trenirani skupa podataka bit će promijenjene za mali nasumični horizontalni i vertikalni pomak (10%) te će bit napravljeno nasumično horizontalno zrcaljenje slika. U obe baze podataka pikseli će biti skalirani n aisti način kao i do sada.

```
# create data generators
train_datagen = ImageDataGenerator(rescale=1.0/255.0,
  width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)

# prepare iterators
train_it = train_datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
  class_mode='binary', batch_size=64, target_size=(200, 200))
test_it = test_datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
  class_mode='binary', batch_size=64, target_size=(200, 200))
```

Slika 5.10 primjena proširivanja treniranog skupa podataka

Rezultati koje smo dobili u ovom slučaju prikazani su u sljedećoj tablici:

Table 5.4 Rezultati nakon optimizacije metodom *Data agumentation*

3 VGG BLOK		Optimizacija: SGD Točnost: 85.769 % Epohe: 50
3 VGG BLOK		Optimizacija: SGD Točnost: 80,819 % Epohe: 50

Analizom krivulja za učenje model je podložan daljnjem učenju gdje se gubitak smanjiva i nad treniranom i testnom bazom podataka nakon 50 ponavljanja. Ukoliko poduplamo broj epoha na

100 postoji velika vjerojatnost da će model rezultirati boljom klasifikacijom. Umjesto horizontalnog zrcaljena pokušala se koristiti promjena svjetline slika i njihova rotacija međutim dobila se manja klasifkacijska točnost s nešto lošijim krivuljama učenja.

Kao što je i pretpostavljeno na samom početku korištenjem raznih metoda regularizacije i optimizacije smanjiva se prekomjerno prilagođavanje i usporava se progresija algoritma za učenje što za posljedicu ima poboljšane rada naše konvolucijske neuronske mreže. Odabrani osnovni model nudi veći kapacitet nego što je zapravo potrebno za ovaj problem pa bi valjalo istražiti alternativne arhitekture. Manji model će biti brži za treniranje te bi mogao rezultirati boljim izvedbom klasifikacije.

5.5. Preneseno učenje

Preneseno učenje nam omogućava da iskoristimo već unaprijed istrenirane modele neuronskih mreža te ih primijenimo na svojoj bazi podataka. U nastavku ćemo analizirati preneseno učenje koristeći se VGG-16 modelom, koji je imao veliku točnost klasifikacije na *ImageNet* bazi podataka. Model se sastoji od dijela za ekstrakciju značajki i potpuno povezanih konvolucijskih slojeva.[23]

Kako bi prilagodili model problemu klasifikacije baze slika pasa i mačaka zadržat ćemo dio za ekstrakciju značajki ta na njega dodati svoje prilagođene klasifikatore. To znači da ćemo zadržati trenirane težine svih konvolucijskih slojeva nepromjenjivim tokom novog treniranja, a treniranje ćemo vršiti samo na potpuno povezanim slojevima. Cilj tog sloja će biti naučiti interpretirati značajke iz modela te na osnovu njih izraditi binarnu klasifikaciju.[23]

```
def define_model():
  # load model
  model = VGG16(include_top=False, input_shape=(224, 224, 3))
  # mark loaded layers as not trainable
  for layer in model.layers:
    layer.trainable = False
  # add new classifier layers
  flat1 = Flatten()(model.layers[-1].output)
  class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
  output = Dense(1, activation='sigmoid')(class1)
  # define new model
  model = Model(inputs=model.inputs, outputs=output)
  # compile model
  opt = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False)
  model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

Slika 5.11 Prilagođeni VGG-16 model

Kako je i prikazano na Slika 5.11 ukinuli smo potpuno povezani sloj s kraja modela VGG-16 što znači da će on završiti zadnjim maksimalnim sabirnim slojem. Kreiranjem novog izlaznog sloja koji se sadrži nove prilagođene klasifikatore te poravnavajući sloj možemo krenuti u treniranje mreže.

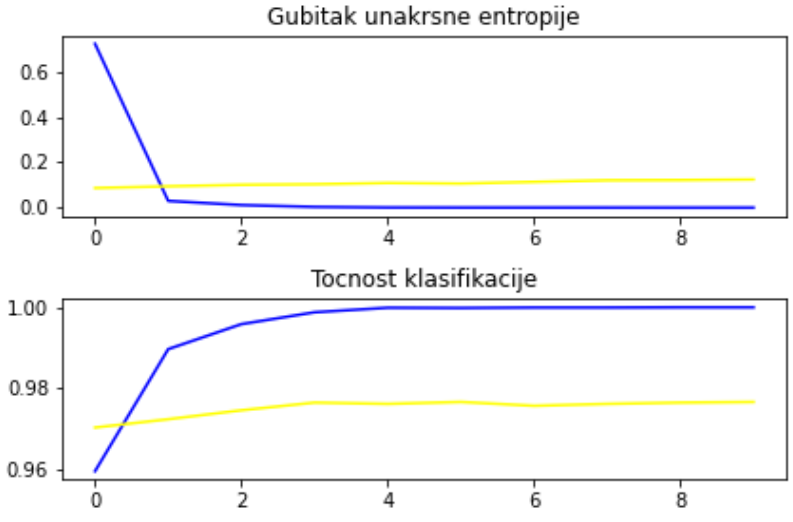
Budući da trebamo istrenirati samo novi nadodani izlazni sloj, cijeli proces učenja će trajati puno kraće nego u prethodnim primjerima. Broj epoha ćemo postaviti na 10. Za VGG-3 model ciljana veličina slika je različita od one u modelu VGG-16 pa će se morati prilagodit te će se pikseli morati centrirati.

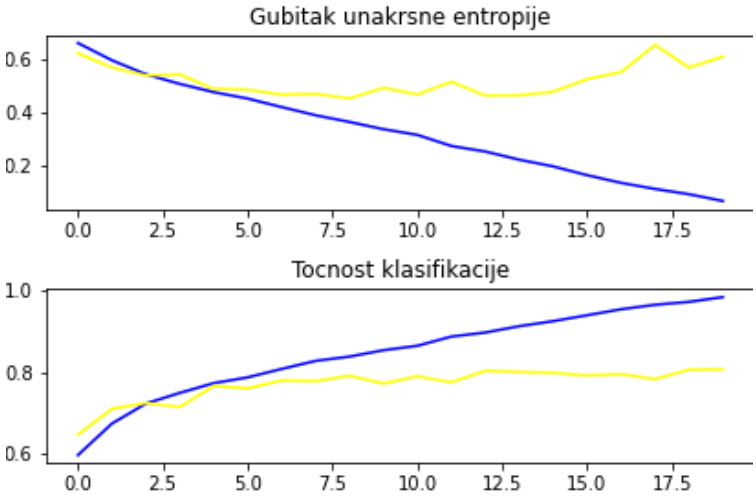
```
# create data generator
datagen = ImageDataGenerator(featurewise_center=True)
# specify imagenet mean values for centering
datagen.mean = [123.68, 116.779, 103.939]
# prepare iterator
train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
  class_mode='binary', batch_size=64, target_size=(224, 224))
test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
  class_mode='binary', batch_size=64, target_size=(224, 224))
```

Slika 5.12 Centraliziranje piksela i dimenzioniranje slika

Usporedba osnovnog modela VGG-3 I VGG-16 dana je tablicom u nastavku.

Table 5.5 Usporedba prenessenog učenja i osnovnog modela

VGG-16 Preneseno učenje		Optimizacija: adam Točnost: 97.652%

3 VGG BLOKA		Optimizacija: SGD Točnost: 80.739%

Analizom krivulja učenja možemo vidjeti da model vrlo brzo opisuje bazu podataka te ne pokazuje znakove *overfittinga*, dok kod osnovnog modela javlja već uranim fazama treniranja. Shodno s rezultatima ukazuje se prostor za uvođenjem dodatnog kapaciteta u klasifikatoru uz kombinaciju regularizacije s nekom od prije navedenih metoda. Puno brži i točniji prooces treniranja modela od osnovnog. Prerano *overfittanja* osnovnog modela znači da model prepoznaje specifične slike iz trenirane baze umjesto da prepozna generalne značajke koje može primijeniti na novim primjerima koji se nalaze u testnoj bazi. S tim njegovo učenje nakon 5 epohe postaje beznačajno jer se ne može iskoristit.

Ukoliko smo zadovoljni konfiguracijom svog modela možemo je istrenirati na cjelovitoj bazi podataka potom spremiti i koristit za klasifikaciju pojedinačne slike kao ulaza.

Zbog jednostavnosti cjeloviti kod svih primjera navedenih u ovom radu prikazani su u dodatku u PDF formatu te ga se može koristiti kao refernca prilikom čitanja.

ZAKLJUČAK

Konvolucijske neuronske mreže predstavljaju jedna od najčešćih arhitektura dubokog učenja, a najčešće se primjenjuje za analizu računalnog vida. Svoj veliki procvat su doživjele pojavom novih GPU kartica koju su omogućile veliku brzinu izvedbe modela neuronskih mreža.

Kroz ovaj rad sam se upoznala s osnovama razvijanja konvolucijske mreže, njezinim slojevima te mogućim problemima s kojim se susreće algoritam prilikom vršenja matematičkih operacija nad zadanim podacima.

Cilj ovog rada je bio implementirati osnovne ideje razvoja mreže te usporediti izvedbe raznih varijacija na osnovni model. Glavna zadaća razvijene konovlucijske neuronske mreže je bila binarna klasifikacija nad zadanim skupom koji je sadržava 25000 slika pasa i mačaka. Kako bi se pravilno postavio model i konačni problem bio rješiv uvela se pretpostavka da se na svakoj slici nalazi samo jedna mačka ili pas. Baza podataka se morala pripremit na način da se dio slika odvoji u testni skup, a dio u skup za treniranje. Nakon razvijanja osnovnog VGG modela slaganjem blokova procjenjivala se točnost klasifikacije mreže. Nadalje se kroz grafove učenja se analizirala sposobnost naše mreže tijekom treniranja da generalizira svojstva klasifikacije te mogućnost primjene istih na neviđenom skupu podataka. Dodavanjem dodatnih blokova točnost klasifikacije se povećavala no problem *overfittanja* se pojavljivo sve ranije. Primjenom metoda optimizacije i regulacije pokazano je kako ukloniti taj problem te postavljajući vrijednosti parametara na različite vrijednosti pratio se performans mreže. Kao najboljom metodom se pokazalo preneseno učenje koje koristi unaprijed istrenirane mreže (poput VGG-16) na sličnim bazama podataka. Postavljanjem klasifikatora da odgovaraju našem problemu i koristeći se naučenim težinama mogli smo kalsificirat našu početnu bazu slika mačaka i pasa s točnošću od 97,652%.

Početnu kompleksnu ideju s početka rada smo uspijeli realizirati zahvaljući jednostavnosti arhitekture slaganja blokova te znanja koja sam stekla pisajući rad moći ću implementirati i kasnijem nastavku svog školovanja.

LITERATURA

[1] Artificial_neural_network, https://en.wikipedia.org/wiki/Artificial_neural_network#History , Pristupljeno: 1.9.2022.

[2] Džomba K. Konvolucijske neuronske mreže [diplomksi rad]. Zagreb: Prirodoslovno-matematički fakultet matematički odsjek;2018.

[3] Activation Functions in Neural Networks [12 Types & Use Cases], https://www.v7labs.com/blog/neural-networks-activation-functions , Pristupljeno: 1.9.2022.

[4] Krambeger T, Nožica B, Dodig I, Cafuta D. PREGLED TEHNOLOGIJA U NEURONSKIM MREŽAMA. Tehničko veleučilište u Zagrebu. Vol. 7, No. 1, 2019. DOI: 10.19279/TVZ.PD.2019-7-1-04

[5] Loss Functions and Their Use In Neural Networks, https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9 , Pristupljeno: 1.9.2022.

[6] Binary Cross Entropy/Log Loss for Binary Classification, https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/ , Pristupljeno: 1.9.2022.

[7] Židov I. Uvod u neuronske mreže [Završni rad]. Osijek: Sveučilišni preddiplomski studij matematike; 2018.

[8] A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 , Pristupljeno: 4.9.2022.

[9] Dilberović I. Prepoznavanje slika pomoću kovolucijske neuronske mreže [završni rad]. Zagreb:Visoko učilište algebra; 2020.

[10] Vukotić V. Raspoznavanje objekata dubokim neuronskim mrežama [diplomksi rad]. Zagreb: Fakultet elektrotehnike i računarstav; 2014.

[11] Khan S., Rahmani H., Salah S.A.A., Bennamoun M. A Guide to Convolutional Neural Networks for Computer Vision. Toronto: Morgan & Claypool Publishers; 2018.

[12] A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 , Pristupljeno: 4.9.2022.

[13] Weight Initialization for Deep Learning Neural Networks, https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/, Pristupljeno: 5.9.2022.

[14] Why cautiously initializing deep neural networks matters?, https://towardsdatascience.com/what-is-weight-initialization-in-neural-nets-and-why-it-matters-ec45398f99fa , Pristupljeno: 5.9.2022.

[15] A Gentle Introduction to Batch Normalization for Deep Neural Networks, https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/ , Pristupljeno: 5.9.2022.

[16] Simonyan K., Zisserman A. Netwirks for large-scale image recognition. Published as a conference paper at ICLR 2015 VERY DEEP CONVOLUTIONAL; Oxford, Engleska, 10.4.2015. Oxford: University of Oxford; 2015.

[17] VGG16 – Convolutional Network for Classification and Detection, https://neurohive.io/en/popular-networks/vgg16/ , Pristupljeno: 7.9.2022.

[18] Google Colab for Machine Learning Projects, https://machinelearningmastery.com/google-colab-for-machine-learning-projects/ , Pristupljeno:7.9.2022.

[19] Wikipedia – Keras, https://en.wikipedia.org/wiki/Keras , Pristupljeno:7.9.2022.

[20] Keras, https://keras.io/why_keras/ , Pristupljeno:8.9.2022.

[21] Abadi, Martín; Barham, Paul; Chen, Jianmin; Chen, Zhifeng; Davis, Andy; Dean, Jeffrey; Devin, Matthieu; Ghemawat, Sanjay; Irving, Geoffrey; Isard, Michael; Kudlur, Manjunath; Levenberg, Josh; Monga, Rajat; Moore, Sherry; Murray, Derek G.; Steiner, Benoit; Tucker, Paul; Vasudevan, Vijay; Warden, Pete; Wicke, Martin; Yu, Yuan; Zheng, Xiaoqiang. TensorFlow: A System for Large-Scale Machine Learning. Open access to the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation is sponsored by USENIX. Savannah, GA, USA, 2.-4.11.2016.

[22] How to classify Photos of Dogs and Cats (with 97%accuracy) https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/ , Pristupljeno:8.9.2022.

[23] How to Classify Photos of Dogs and Cats (with 97% accuracy), https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/ , Pristupljeno:8.9.2022

[24] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ , Pristupljeno: 9.9.2022

PRILOZI

I. Python skripta s potpunim kodom

POTPUNI KOD ZA DANI PROBLEM

```
#POVEZIVANJE GOOGLE DRIVE-A I COLABA
from google.colab import drive
drive.mount('/content/drive')


#UČITAVANJE BAZE PODATAKA IZ ZIP DATOTEKE
import zipfile
import os

zip_ref = zipfile.ZipFile('/content/drive/MyDrive/train.zip', 'r') #otvranjae zip file
za citanje
zip_ref.extractall('/tmp') #vadi datoteke u /temp folder
zip_ref.close()


# PRIKAZ NESREĐENE STRUKTURE SLIKA
from matplotlib import pyplot
from matplotlib.image import imread


# lokacija moje baze podataka
folder = '/tmp/train/'

# prikaz par slika iz baze podataka
for i in range(9):
        # define subplot
        pyplot.subplot(330 + 1 + i)
        # define filename
        filename = folder + 'cat.' + str(i) + '.jpg'
        # load image pixels
        image = imread(filename)
        # plot raw pixel data
        pyplot.imshow(image)
# ispiši graf
pyplot.show()

# POTREBNE BIBLIOTEKE

from os import makedirs
from os import listdir
from shutil import copyfile
from random import seed
from random import random

#ORGANIZIRANJE BAZE PODATAKA U POTREBNU STRUKTURU

# STVARANJA DIREKTORIJA
dataset_home = 'dataset_dogs_vs_cats/'
subdirs = ['train/', 'test/']
for subdir in subdirs:
        # imenovanje labela
        labeldirs = ['dogs/', 'cats/']
        for labldir in labeldirs:
```

```
                newdir = dataset_home + subdir + labldir
                makedirs(newdir, exist_ok=True)

# generator nasumičnog broja
seed(1)

# definicija omjera za slike koje će se korisitit za validaciju
val_ratio = 0.25


# kopiranje slika u pripadjuće direktorije
src_directory = '/tmp/train/'
for file in listdir(src_directory):
        src = src_directory + '/' + file
        dst_dir = 'train/'
        if random() < val_ratio:
                dst_dir = 'test/'
        if file.startswith('cat'):
                dst = dataset_home + dst_dir + 'cats/'  + file
                copyfile(src, dst)
        elif file.startswith('dog'):
                dst = dataset_home + dst_dir + 'dogs/'  + file
                copyfile(src, dst)

# PROVJERA RASPODJELE U DIREKTORIJE

for i in range (4):
  a = len(os.listdir('dataset_dogs_vs_cats/train/cats/'))

  b = len(os.listdir('dataset_dogs_vs_cats/test/cats/'))

  c = len(os.listdir('dataset_dogs_vs_cats/train/dogs/'))

  d = len(os.listdir('dataset_dogs_vs_cats/test/dogs/'))
print(a,b,c,d)



# OSNOVNI MODEL S JEDNIM BLOKOM

from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
import sys
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator


# definiranje pomoću jednog bloka VGG strukture
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
```

```
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = SGD(learning_rate=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# grafovi krivulja ucenja
def summarize_diagnostics(history):
        # plot gubitak unakrsne entropije
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot tocnosti
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # spremanje grafova
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_1plot.png')
        pyplot.close()


# EVALUACIJA MODELA

def run_test_harness():
        # definicija modela
        model = define_model()

        # definiran data generator
        datagen = ImageDataGenerator(rescale=1.0/255.0)

        # priprema iteratora
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))

        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=20,
verbose=1)

        # procjena modela
        _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=1)
        print('> %.3f' % (acc * 100.0))

        # krivulja učenja
        summarize_diagnostics(history)

# ulazna točka, pokreni test
run_test_harness()
```

```
# OSNOVNI MODEL S DVA BLOA

from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
import sys
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator


# definiranje pomoću jednog bloka VGG strukture
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = SGD(learning_rate=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# grafovi krivulja ucenja
def summarize_diagnostics(history):
        # plot gubitak unakrsne entropije
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot tocnosti
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # spremanje grafova
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_1plot.png')
        pyplot.close()


# EVALUACIJA MODELA

def run_test_harness():
        # definicija modela
```

```
        model = define_model()

        # definiran data generator
        datagen = ImageDataGenerator(rescale=1.0/255.0)

        # priprema iteratora
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))

        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=20,
verbose=1)

        # procjena modela
        _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=1)
        print('> %.3f' % (acc * 100.0))

        # krivulja učenja
        summarize_diagnostics(history)

# ulazna točka, pokreni test
run_test_harness()



# OSNOVNI MODEL S TRI BLOKA

from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
import sys
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator


# definiranje pomoću jednog bloka VGG strukture
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))
```

```
        # compile model
        opt = SGD(learning_rate=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# grafovi krivulja ucenja
def summarize_diagnostics(history):
        # plot gubitak unakrsne entropije
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot tocnosti
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # spremanje grafova
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_1plot.png')
        pyplot.close()


# EVALUACIJA MODELA

def run_test_harness():
        # definicija modela
        model = define_model()

        # definiran data generator
        datagen = ImageDataGenerator(rescale=1.0/255.0)

        # priprema iteratora
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))

        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=20,
verbose=1)

        # procjena modela
        _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=1)
        print('> %.3f' % (acc * 100.0))

        # krivulja učenja
        summarize_diagnostics(history)

# ulazna točka, pokreni test
run_test_harness()



# OSNOVNI MODEL VGG STRUKTURE S 3 BLOKA UZ ADAM OPTIMIZATOR
```

```
import sys
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam


# definiranje pomoću TRI bloka VGG strukture
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))

        # compile model
        opt =Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# grafovi krivulja ucenja
def summarize_diagnostics(history):
        # plot gubitak unakrsne entropije
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot tocnosti
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # spremanje grafova
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_3plot.png')
        pyplot.close()


# EVALUACIJA I TRENIRANJE MODELA

def run_test_harness():
        # definiranje modela
```

```
        model = define_model()

        # create data generator
        datagen = ImageDataGenerator(rescale=1.0/255.0)

        # iteratori
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))

        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=20,
verbose=1)

        # evaluate model
        _, acc = model.evaluate(test_it, steps=len(test_it), verbose=1)
        print('> %.3f' % (acc * 100.0))
        # krivulje učenja
        summarize_diagnostics(history)

# ulazna točka, pokreće test
run_test_harness()

# OSNOVNI MODEL VGG STRUKTURE S 3 BLOKA UZ ADAM OPTIMIZATOR
import sys
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam


# definiranje pomoću TRI bloka VGG strukture
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))

        # compile model
        opt =Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```

```
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# grafovi krivulja ucenja
def summarize_diagnostics(history):
        # plot gubitak unakrsne entropije
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot tocnosti
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # spremanje grafova
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_3plot.png')
        pyplot.close()


# EVALUACIJA I TRENIRANJE MODELA

def run_test_harness():
        # definiranje modela
        model = define_model()

        # create data generator
        datagen = ImageDataGenerator(rescale=1.0/255.0)

        # iteratori
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))

        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=40,
verbose=1)

        # evaluate model
        _, acc = model.evaluate(test_it, steps=len(test_it), verbose=1)
        print('> %.3f' % (acc * 100.0))
        # krivulje učenja
        summarize_diagnostics(history)

# ulazna točka, pokreće test
run_test_harness()



# OSNOVNI MODEL VGG STRUKTURE S 3 BLOKA UZ ADAM OPTIMIZATOR I DROPOUT REGULARIZACIJU
import sys
from matplotlib import pyplot
from keras.models import Sequential
```

```
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from keras.backend_config import epsilon


# definiranje pomoću TRI bloka VGG strukture, adam, dropout
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform',padding='same',input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.2))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.3))
        model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.4))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dropout(0.5))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# grafovi krivulja ucenja
def summarize_diagnostics(history):
        # plot gubitak unakrsne entropije
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot tocnosti
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # spremanje grafova
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_3_40plot.png')
        pyplot.close()


# run the test harness for evaluating a model
def run_test_harness():
```

```
        # define model
        model = define_model()
        # create data generator
        datagen = ImageDataGenerator(rescale=1.0/255.0)
        # prepare iterators
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=40,
verbose=0)
        # evaluate model
        _, acc = model.evaluate(test_it, steps=len(test_it), verbose=0)
        print('> %.3f' % (acc * 100.0))
        # learning curves
        summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()



# OSNOVNI MODEL VGG STRUKTURE S 3 BLOKA UZ SGD OPTIMIZATOR I DROPOUT REGULARIZACIJU

import sys
from matplotlib import pyplot
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from keras.backend_config import epsilon

# definiranje pomoću TRI bloka VGG strukture, sgd, dropout
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.2))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.3))
        model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.4))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
```

```
        model.add(Dropout(0.5))
        model.add(Dense(1, activation='sigmoid'))

        # compile model
        opt = SGD(learning_rate=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])

        return model

# grafovi krivulja ucenja

def summarize_diagnostics(history):
        # plot gubitak unakrsne entropije
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot tocnosti
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # spremanje grafova
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_3plot.png')
        pyplot.close()


# run the test harness for evaluating a model

def run_test_harness():
        # define model
        model = define_model()
        # create data generator
        datagen = ImageDataGenerator(rescale=1.0/255.0)
        # prepare iterators
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=20,
verbose=0)
        # evaluate model
        _, acc = model.evaluate(test_it, steps=len(test_it), verbose=0)
        print('> %.3f' % (acc * 100.0))
        # learning curves
        summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()



# DATA AGUMENTATION
```

```
# OSNOVNI MODEL VGG STRUKTURE S 3 BLOKA UZ ADAM OPTIMIZATOR I DROPOUT REGULARIZACIJU

import sys
from matplotlib import pyplot
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from keras.backend_config import epsilon

# define cnn model
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))

        # compile model
        opt = SGD(lr=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
        # plot loss
        pyplot.subplot(211)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='orange', label='test')
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Classification Accuracy')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='orange', label='test')
        # save plot to file
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_data_agu_plot.png')
        pyplot.close()

# EVALUACIJA

def run_test_harness():
```

```
        # define model
        model = define_model()
        # create data generators
        train_datagen = ImageDataGenerator(rescale=1.0/255.0,
                width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
        test_datagen = ImageDataGenerator(rescale=1.0/255.0)
        # prepare iterators
        train_it = train_datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = test_datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=50,
verbose=1)
        # evaluate model
        _, acc = model.evaluate(test_it, steps=len(test_it), verbose=1)
        print('> %.3f' % (acc * 100.0))
        # learning curves
        summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()

# DATA AGUMENTATION, SGD
# baseline model with data augmentation for the dogs vs cats dataset
import sys
from matplotlib import pyplot
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from keras.backend_config import epsilon

# define cnn model
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = SGD(lr=0.001, momentum=0.9)
```

```
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
        # plot loss
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # save plot to file
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_data_agu_plot.png')
        pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
        # define model
        model = define_model()

        # create data generators
        train_datagen = ImageDataGenerator(rescale=1.0/255.0,
                width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
        test_datagen = ImageDataGenerator(rescale=1.0/255.0)

        # prepare iterators
        train_it = train_datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = test_datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))

        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=50,
verbose=1)
        # evaluate model
        _, acc = model.evaluate(test_it, steps=len(test_it), verbose=1)
        print('> %.3f' % (acc * 100.0))
        # learning curves
        summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()



# DATA AGUMENTATION, rotacija i svjetlina
# baseline model with data augmentation for the dogs vs cats dataset
import sys
from matplotlib import pyplot
from tensorflow.keras.utils import to_categorical
```

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from keras.backend_config import epsilon

# define cnn model
def define_model():
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = SGD(lr=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
        # plot loss
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # save plot to file
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_data_agu_rotacija_plot.png')
        pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
        # define model
        model = define_model()
        # create data generators
        train_datagen = ImageDataGenerator(rescale=1.0/255.0,
                width_shift_range=0.1, height_shift_range=0.1,
brightness_range=[0.2,1.0], rotation_range=90)
```

```
        test_datagen = ImageDataGenerator(rescale=1.0/255.0)
        # prepare iterators
        train_it = train_datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = test_datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it), epochs=50,
verbose=1)
        # evaluate model
        _, acc = model.evaluate(test_it, steps=len(test_it), verbose=1)
        print('> %.3f' % (acc * 100.0))
        # learning curves
        summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()

#PRENESENO UČENJE



# ORGANIZACIJA BAZE PODATAKA TAKO DA BUDE PRIGODNA ZA VGG-16 MODEL

from os import makedirs
from os import listdir
from shutil import copyfile
from distutils.dir_util import copy_tree

# create directories
dataset_home = 'finalize_dogs_vs_cats/'
# create label subdirectories
labeldirs = ['dogs/', 'cats/']
for labldir in labeldirs:
        newdir = dataset_home + labldir
        makedirs(newdir, exist_ok=True)
# copy training dataset images into subdirectories
src_directory = 'dataset_dogs_vs_cats/train'
for file in listdir(src_directory):
        src = src_directory + '/' + file
        if file.startswith('cat'):
                dst = dataset_home + 'cats/'
                copy_tree(src, dst)
        elif file.startswith('dog'):
                dst = dataset_home + 'dogs/'
                copy_tree(src, dst)



import sys
from matplotlib import pyplot
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator
```

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from keras.backend_config import epsilon

# define cnn model
def define_model():
        # load model
        model = VGG16(include_top=False, input_shape=(224, 224, 3))
        # mark loaded layers as not trainable
        for layer in model.layers:
                layer.trainable = False
        # add new classifier layers
        flat1 = Flatten()(model.layers[-1].output)
        class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
        output = Dense(1, activation='sigmoid')(class1)
        # define new model
        model = Model(inputs=model.inputs, outputs=output)
        # compile model
        opt = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07,
amsgrad=False)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
        # plot loss
        pyplot.subplot(211)
        pyplot.title('Gubitak unakrsne entropije')
        pyplot.plot(history.history['loss'], color='blue', label='train')
        pyplot.plot(history.history['val_loss'], color='yellow', label='test')
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Tocnost klasifikacije')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='yellow', label='test')
        pyplot.tight_layout()
        # save plot to file
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_plot.png')
        pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
        # define model
        model = define_model()
        # create data generator
        datagen = ImageDataGenerator(featurewise_center=True)
        # specify imagenet mean values for centering
        datagen.mean = [123.68, 116.779, 103.939]
        # prepare iterator
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(224, 224))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(224, 224))
        # fit model
        history = model.fit(train_it, steps_per_epoch=len(train_it),
```

```
                validation_data=test_it, validation_steps=len(test_it), epochs=10,
verbose=1)
        # evaluate model
        _, acc = model.evaluate(test_it, steps=len(test_it), verbose=0)
        print('> %.3f' % (acc * 100.0))
        # learning curves
        summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()



# SPREMANJE ZAVRŠNOG MODELA

from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from keras.backend_config import epsilon


# define cnn model
def define_model():
        # load model
        model = VGG16(include_top=False, input_shape=(224, 224, 3))
        # mark loaded layers as not trainable
        for layer in model.layers:
                layer.trainable = False
        # add new classifier layers
        flat1 = Flatten()(model.layers[-1].output)
        class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
        output = Dense(1, activation='sigmoid')(class1)
        # define new model
        model = Model(inputs=model.inputs, outputs=output)
        # compile model
        opt =Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07,
amsgrad=False)
        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
        return model

# run the test harness for evaluating a model
def run_test_harness():
# define model
        model = define_model()
        # create data generator
        datagen = ImageDataGenerator(featurewise_center=True)
        # specify imagenet mean values for centering
        datagen.mean = [123.68, 116.779, 103.939]
        # prepare iterator
        train_it = datagen.flow_from_directory('finalize_dogs_vs_cats/',
                class_mode='binary', batch_size=64, target_size=(224, 224))
```

```
        # fit model
        model.fit(train_it, steps_per_epoch=len(train_it), epochs=10, verbose=1)
        # save model
        model.save('final_model_adam.h5')

# entry point, run the test harness
run_test_harness()



# PREDIKCIJA

from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model

# load and prepare the image
def load_image(filename, experimental_relax_shapes=True):
        # load the image
        img = load_img(filename, target_size=(224, 224))
        # convert to array
        img = img_to_array(img)
        # reshape into a single sample with 3 channels
        img = img.reshape(1, 224, 224, 3)
        # center pixel data
        img = img.astype('float32')
        img = img - [123.68, 116.779, 103.939]
        return img

# load an image and predict the class
def run_example():
        # load the image
        img = load_image('/content/dataset_dogs_vs_cats/test/cats/cat.10035.jpg')
        # load model
        model = load_model('final_model.h5')
        # predict the class
        result = model.predict(img)
        print(result[0])

# entry point, run the example
run_example()
```